Titre: Title:	Mesure de la précision des compteurs de cycle et horloges internes des ordinateurs
Auteur: Author:	Hicham Marouani
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Marouani, H. (2004). Mesure de la précision des compteurs de cycle et horloges internes des ordinateurs [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7417/

UNIVERSITÉ DE MONTRÉAL

MESURE DE LA PRÉCISION DES COMPTEURS DE CYCLE ET HORLOGES INTERNES DES ORDINATEURS

HICHAM MAROUANI
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
NOVEMBRE 2004

Your file Votre référence
ISBN: 0-494-01366-4
Our file Notre référence
ISBN: 0-494-01366-4

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MESURE DE LA PRÉCISION DES COMPTEURS DE CYCLE ET HORLOGES INTERNES DES ORDINATEURS

présenté par : MAROUANI Hicham

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen composé de :

M. GUIBAULT François, Ph. D., président

M. DAGENAIS Michel, Ph.D., membre et directeur de recherche

M. GAGNON Michel, Ph.D., membre

À ma femme et mon fils
À mes parents, mes deux soeurs et ma famille
À mes beaux parents

Remerciements

Je tiens à exprimer ma reconnaissance à mon directeur de recherche, le professeur Michel Dagenais. Je le remercie plus particulièrement pour son soutien, sa confiance, sa disponibilité, ses conseils et ses commentaires.

Je désire ensuite remercier Monsieur Benoît Des Ligneris et tous les employés de la compagnie Révolution Linux pour leur coopération.

Je désire également remercier ma famille présente au Canada et aux États-unis d'Amérique ainsi que mes amis pour leurs encouragements et leur soutien.

Résumé

Les grappes d'ordinateurs sont devenues de plus en plus populaires et sont utilisées pour résoudre des problèmes scientifiques complexes. Parmi les outils indispensables pour les utilisateurs d'une grappe de calcul, on note l'outil d'analyse de performance. En effet, une fois que les développeurs ont écrit leurs programmes parallèles, ils doivent faire usage d'un tel outil pour obtenir une analyse de l'exécution de leurs programmes, afin de déceler les lignes de code qu'ils doivent optimiser pour accélérer l'exécution de leurs programmes.

La première version de l'outil d'analyse de performance Linux Trace ToolKit « LTT » était destinée à l'analyse de programmes séquentiels. Dans ce travail, deux composantes essentielles pour la mise en place d'une version parallèle de LTT sont étudiées. Cette étude constitue donc la base essentielle pour le développement d'une version parallèle d'outils d'analyse de performance.

La première composante concerne le système de mesure du temps. LTT peut tracer deux événements qui se sont produits durant l'exécution du programme à analyser dont la différence de temps d'arrivée est de quelques nanosecondes. La résolution de l'horloge logicielle offerte par Linux est de l'ordre de la microseconde, elle ne permet donc pas de tracer avec précision les événements pris en compte par LTT. Ainsi, ce travail vérifie si le registre compteur de cycles horloge microprocesseur peut être utilisé comme système de temps à haute résolution pouvant répondre aux exigences de LTT.

La deuxième composante concerne le problème classique de la synchronisation des horloges dans un système réparti. Pour mieux comprendre le comportement d'un programme parallèle, il faut corréler les événements dans les traces générées par LTT sur les différents nœuds avec précision. Ainsi, il faudra que les horloges des nœuds de la grappe soient synchronisées, ou que leur décalage d'horloge soit connu.

Les solutions de synchronisation sont basées sur un principe probabiliste concernant la symétrie du délai de transfert ou de diffusion des messages de synchronisation entre les nœuds à synchroniser. Ainsi, on se propose d'étudier le temps réel de transfert et de diffusion des messages de synchronisation dans l'architecture réseau la plus utilisée, Ethernet 100 Mbps, afin de déterminer la synchronisation qu'on peut atteindre dans ce genre de réseau.

Les résultats obtenus nous ont conduit à conclure ce qui suit. Premièrement, le compteur de cycles d'horloge du microprocesseur s'avère plus adéquat que l'horloge du système Linux en termes de résolution, tout en maintenant une bonne précision (une résolution d'un cycle, soit typiquement moins d'une nanoseconde, avec une erreur de précision de ± 0.5 microseconde / seconde). Deuxièmement, on peut atteindre une synchronisation des horloges des nœuds dans un système réparti égale à 1.405 ± 3.094 µs en utilisant la technique de diffusion et égale à 10.493 ± 7.015 µs en utilisant la technique de transmission point à point.

Abstract

Computer clusters are increasingly popular to solve complex scientific problems. Among the most important tools for the clusters programmers is the performance analysis tool. Once the developers have written their parallel programs, they have to use such tools to analyze the execution of their programs; hence they can identify the areas to optimize in their source code to shorten the execution time of their programs.

The first version of the performance analysis tool Linux Trace ToolKit « LTT » was built for analyzing sequential programs. In this study, two essential components for setting up a parallel version of LTT are studied.

The first component is the time keeping system. LTT can trace two events that happen within a few nanoseconds of each other. The Linux software clock resolution is about 1 microsecond. Therefore, it cannot accurately trace the two closely spaced LTT events. This study evaluates if the microprocessor clock cycles counter register can be used as a high resolution timing system that can fulfill the needs of LTT.

The second component is the classical problem of clock synchronization in distributed systems. In order to understand the behaviour of a parallel program, the events in traces generated by LTT from different nodes must be accurately correlated. Moreover the cluster nodes clocks must be synchronized, or their offset precisely known.

The synchronization solutions are based on a probabilistic property of the symmetry of the delay of the transmission or of the broadcast of the synchronization messages

between the nodes to synchronize. Thus, we propose to study the transmission and broadcast delay of the synchronization messages in the most used network architecture, Ethernet 100 Mbps, to characterize the synchronization accuracy achievable.

The results lead us to the following conclusions. First, the microprocessor clock cycles counter register is more adequate than the Linux clock system in terms of resolution while maintaining a good accuracy (resolution of at least 1 nanosecond with an error of precision of ± 0.5 microsecond / second). Second, it is possible to synchronize the nodes clocks within 1.405 ± 3.094 μs when using the broadcast messages, and within 10.493 ± 7.015 μs when using point to point messages

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et abréviations

GPS	Global Positioning System
I/O APIC	Input/Output Advanced Programmable Interrupt Controller
IDT	Interrupt Descriptor Table
IRQ	Interrupt Request
LTT	Linux Trace ToolKit
MPI	Message Passing Interface
NTP	Network Time Protocol
PAG	Program Activity Graph
PCC	Processor Clock Cycle
PIC	Programmable Interrupt Controller
PIT	Programmable Interrupt Timer
PPS	Pulse Per Second
RTC	Real Time Clock
SAPIC	Streamlined Advanced Programmable Interrupt Controller
TSC	TimeStamp Counter
UTC	Coordinated Universal Time

Listes des annexes

Introduction

Le temps a toujours été un élément clé dans la vie des humains. En effet, il permet aux humains de garder trace des événements dans un ordre chronologique et ainsi donner sens à notre vie. A travers les différents âges de l'histoire humaine, la granularité de l'unité de mesure de temps a évolué. En fait, la granularité dépend en grande partie de la quantité d'événements qui peuvent survenir dans un laps de temps donné.

Les deux dernières décennies ont été marquées par l'importante évolution des technologies de l'information et plus particulièrement des microprocesseurs. Ces derniers n'ont cessé de gagner en vitesse d'exécution et en performance. Si on prend l'exemple de l'architecture Intel, à la fin des années 1970 les microprocesseurs étaient cadencés à une vitesse de 4,77 MHz, et de nos jours leur vitesse d'exécution frôle les 4 GHz. La plupart des récents microprocesseurs disposent d'un registre qui est incrémenté d'une manière matérielle (sans retard pour le processeur pour sa mise à jour) à chaque cycle d'horloge du microprocesseur. Ce registre est appelé « registre compteur de cycles horloge». Pour un microprocesseur cadencé à 4 GHz, ce registre est incrémenté à chaque 0,25 nanoseconde, une résolution qui est essentielle de nos jours pour baliser la quantité d'événements qui surgissent dans un laps de temps très court.

La majeure partie du présent projet consiste à étudier le registre compteur de cycles d'horloge de différentes architectures (Intel32, Itanium) et de différents constructeurs

(Intel, AMD et VIA) de microprocesseur, afin de déterminer sa précision, sa stabilité et les facteurs qui peuvent influencer son fonctionnement, tels que la variation de la température. Ceci permettra de voir jusqu'à quel point ce registre peut être utilisé comme une source de mesure de temps à haute résolution et précision, afin de comparer chronologiquement les événements provenant de plusieurs traces dans une grappe d'ordinateurs.

L'objectif à plus long terme est d'élaborer une version parallèle de l'outil d'analyse de performance « Linux Trace Toolkit (LTT) » [1].

Le contexte cible est le projet MAMMOUTH [2] qui consiste en la construction d'une grappe de calcul de plus de 1024 nœuds et qui doit constituer le futur centre de calcul scientifique de l'université de Sherbrooke. Ce dernier sera destiné à usage académique ainsi que commercial, pour effectuer des calculs scientifiques. Offrir le super ordinateur seul n'est pas suffisant pour les utilisateurs, il faudra l'accompagner d'outils adéquats, tel que compilateurs (C, FORTRAN..), librairies de programmation parallèle (MPI), outils d'analyse de performance, et d'autres services comme le travail à distance. L'outil d'analyse de performance s'avère d'une importance particulière, car il ne suffit pas de développer un programme parallèle qui permette d'effectuer des calculs scientifiques, il faudra un outil d'analyse de performance parallèle qui fournit une analyse de l'exécution du programme, afin que le développeur puisse déceler les lignes de code qu'il doit optimiser pour accélérer son programme. Ceci explique l'idée de développer une version parallèle de l'outil d'analyse de performance LTT, permettant d'analyser la performance des programmes parallèles. La version originale de LTT a été réalisée par Karim

Yaghmour dans le laboratoire CASI (Conception et Analyse des Systèmes Informatique) de l'école Polytechnique de Montréal, sous la direction du professeur M. Michel Dagenais.

Un programme de calcul scientifique est un programme parallèle, et souvent un programme réparti. Au moment de l'exécution du programme parallèle, des instances du programme sont donc exécutées sur plusieurs nœuds, puis elles entrent en communication afin de diviser la tâche de calcul et terminer le calcul dans un délai plus court. Dans sa version actuelle, LTT génère la trace d'exécution d'un programme s'exécutant sur un seul ordinateur. Donc, le premier réflexe ou la solution abstraite du projet consiste à faire exécuter LTT sur tous les nœuds où le programme parallèle s'exécute, puis à analyser les traces générées par LTT afin d'optimiser le programme parallèle. C'est ici que les difficultés commencent et motivent le présent projet.

LTT permet de tracer deux événements dont la différence de temps d'arrivée est de quelques nanosecondes (e.g. des données sont parvenues au port série, l'interruption correspondante sera activée, juste après des données arrivent sur la carte réseau, donc l'interruption de la carte réseau sera activée). Le problème est que la résolution de l'horloge logicielle du système Linux est de l'ordre de la microseconde, donc on ne peut pas ordonnancer avec exactitude les deux événements cités dans l'exemple précédent. D'où la nécessité d'étudier le registre compteur de cycles d'horloge qui offre une résolution de l'ordre de la nanoseconde, afin de savoir jusqu'à quel point il peut être utilisé comme système de mesure de temps pouvant répondre aux exigences de LTT. Bien que ce registre soit actuellement utilisé dans la récente version de LTT, et qu'il soit

utilisé par le module de maintien de temps du noyau de Linux pour interpoler le temps, il reste à caractériser sa précision et sa stabilité en vue d'une utilisation en réparti. Notons bien que l'idée est de ne pas faire appel à du matériel externe coûteux pour avoir un système de temps à haute résolution et précision.

Pour donner une vue exacte de l'exécution d'un programme parallèle sur différents nœuds, il faut que ces derniers soient synchronisés dans le temps afin de bien corréler les traces des différents nœuds. C'est le problème classique des systèmes parallèles (répartis) et qui est la synchronisation des horloges. En effet, les ordinateurs sont munis d'horloges pour maintenir le temps et elles sont sujettes à des déviations au cours du temps. En plus, au moment t0, les horloges de la grappe ne sont pas synchronisées. La solution la plus familière est NTP[3][4]. Elle est basée sur le principe de probabilité pour l'estimation du temps d'échange de message entre deux nœuds. L'erreur de précision de cette dernière dépend de l'architecture et de l'état du réseau sous lequel le programme NTP tourne. Par exemple, dans un réseau Ethernet 100 Mbps avec accès à un serveur NTP primaire, l'erreur de précision de NTP [21] est autour de 1 milliseconde. Pour améliorer le degré de synchronisation de notre système, on va étudier la symétrie du délai de transfert de messages entre deux nœuds dans un réseau Ethernet (technologie la plus utilisée dans les grappes de calculs scientifiques) afin de déterminer la précision maximale qu'on peut atteindre dans ce type de réseau.

Le développement d'une version parallèle de LTT permettant de générer et de corréler les traces d'exécution du programme parallèle pose plusieurs défis pour rendre l'outil convivial et vraiment utile. Ce genre de programmes tourne sur plusieurs nœuds et on

peut imaginer la complexité d'analyse de toutes ces traces pour déterminer les points qui empêchent le programme de s'exécuter plus vite. C'est dans cette optique que des méthodes d'analyse spécifiques à ce genre de problème seront étudiées.

C'est autour des points mentionnés ci-dessus que le sujet de recherche présenté ici s'articule. Ainsi, le chapitre 1 effectue une revue de littérature concernant le registre compteur de cycles d'horloge, la synchronisation des horloges dans un système réparti et les méthodes d'analyse de performance d'un programme parallèle. Le chapitre 2 décrit les concepts permettant l'étude du registre compteur de cycles d'horloge et la variabilité du temps de transmission réseau. Le chapitre 3 présente les différents algorithmes et structures de données pour l'étude du registre compteur de cycles d'horloge et de la variabilité du temps de transmission réseau. Et en dernier lieu, le chapitre 4 discute les résultats obtenus.

Chapitre 1

Revue de littérature

Ce chapitre illustre les différentes solutions proposées sur les sujets autour desquels s'articule le projet de recherche présenté dans ce mémoire.

Certaines méthodes d'analyse des traces de programmes parallèles sont énumérées au début du chapitre. Puis, une discussion sur les horloges et la proposition de l'étude d'une horloge à haute résolution suit. Finalement, sont présentées des solutions pour la synchronisation des horloges dans un système réparti, et la proposition de l'étude de certains facteurs, pour mieux optimiser les solutions à ce problème.

1.1 Outil d'analyse de performance

Le domaine des technologies de l'information a connu un essor phénoménal. Parmi les facteurs importants de cette avancée, on note les outils d'analyse de performance. La

disponibilité de tels outils est primordiale, autant pour comparer la performance de produits que pour localiser les points faibles de systèmes informatiques, afin de les modifier et les rendre plus performants.

1.1.1 Définition

L'outil d'analyse de performance permet de donner une vue reflétant l'évolution de l'exécution d'un programme et parfois même celle du système sous-jacent, selon l'outil d'analyse de performance utilisé. Pour y parvenir, il s'arrange pour générer un ensemble de données décrivant le déroulement de l'exécution du programme à un moment donné (appel système, appel de fonction, interruption...). La granularité des événements tracés dépend de l'outil utilisé. Certains outils prennent en compte des événements de bas niveau (interruption, appel système), tandis que d'autres s'intéressent seulement aux événements de haut niveau (saut conditionnel, appel de fonction).

Il existe une panoplie d'outils d'analyse de performance. Cependant, ils diffèrent par la manière avec laquelle ils récoltent les données et par le type d'information qu'ils engendrent.

Les différentes manières [5] utilisées pour récolter les données sont les suivantes :

1. Par échantillonnage : méthode qui consiste à interrompre l'exécution du programme périodiquement, afin de prélever les informations concernant l'état de son exécution. De plus en plus, des périodes d'échantillonnage variables sont

utilisées, à l'instar des périodes fixes, afin d'augmenter la précision des résultats obtenus.

2. Par instrumentation : méthode intrusive qui consiste à insérer des points de mesures, soit au niveau du code exécutable, soit au niveau du code source, afin d'obtenir les informations voulues durant l'exécution du programme. La modification du code source se fait par l'insertion des points de mesures dans le code source du programme. L'exemple le plus simple, utilisé par la majorité des programmeurs, est l'ajout de la fonction printf() aux endroits du programme à observer. La modification du code exécutable consiste à utiliser un éditeur de code exécutable, à insérer les points de mesure, et enfin à réécrire le nouvel exécutable.
3. Par simulation : technique qui simule le matériel sur lequel le programme est censé s'exécuter. Ainsi l'exécution du programme sera observée par le simulateur. Son avantage est que l'exécution du programme sera observée sans aucune intrusion ou instrumentation. Par contre, l'inconvénient réside dans le fait de s'assurer que le simulateur dispose de toutes les fonctionnalités du matériel simulé sans trop ralentir l'exécution du programme observé.

Les informations recueillies sont classifiées en deux catégories. La première catégorie, qui est dite information locale, sert à la compréhension d'une composante logicielle isolée. Par contre, la deuxième catégorie, qui est dite information globale, sert à la compréhension de l'interaction entre les diverses composantes logicielles.

Selon le type des données tracées, on peut distinguer différents types d'outil d'analyse de performance [5] et qui sont :

- Analyse locale qui sert à la compréhension d'une composante logicielle isolée;
- Analyse globale qui sert à la compréhension de l'interaction entre les diverses composantes logicielles (e.g. l'utilitaire *top* sous Unix);
- Analyse dynamique qui est basé sur l'instrumentation du système d'exploitation afin d'obtenir l'évolution de l'exécution du programme par rapport au restant du système;
- Simulateur qui simule le matériel sur lequel le programme est censé s'exécuter.

Dans [5], des outils d'analyse de performance réels de chaque type sont présentés et comparés.

1.1.2 Outil d'analyse de performance parallèle des architectures parallèles

Le terme "parallèle" indique que l'outil est conçu pour l'analyse de performance des programmes parallèles. Ces derniers sont des programmes pour résoudre des problèmes complexes, majoritairement scientifiques du genre : simulation des conditions météorologiques, imagerie de synthèse, mécanique des fluides, cryptographie (bris d'un code)....

Ce genre de problème ne peut pas être traité par un programme séquentiel, en utilisant les ressources d'un seul processeur. Ainsi, un programme parallèle est codé de sorte qu'il puisse s'exécuter sur plusieurs processeurs (ordinateur multiprocesseurs, différents

nœuds en réseau ou la combinaison des deux). Donc, il dispose de la puissance de calcul de tous ces derniers, grâce à des librairies de programmation comme MPI [22] (Message Passing Interface), qui permettent aux instances du programme de communiquer entre elles et de coopérer pour résoudre le problème.

Le principe de base de l'outil d'analyse de performance parallèle est similaire à celui de l'outil d'analyse de performance conventionnel parce que l'exécution d'un programme parallèle consiste au clonage du programme exécutable sur plusieurs nœuds. Donc, il suffit d'observer l'exécution de chaque copie du programme sur le nœud où elle s'exécute, afin de collecter les informations appropriées sur l'exécution.

En premier lieu, if faut trouver le moyen de synchroniser dans le temps ces informations, car les horloges dans un système réparti sont sujettes à la désynchronisation. Ceci nous permet d'avoir une vue cohérente de l'exécution du programme parallèle sur les différents nœuds et former ce qu'on appelle le graphe d'activités du programme PAG.

En deuxième lieu, if faut offrir à l'utilisateur le moyen d'analyser le PAG, afin de le guider vers les parties de son programme qui doivent être optimisées, afin que le temps d'exécution du programme soit diminué.

1.1.3 Graphe d'activités du programme (PAG)

Une fois les traces du programme parallèle obtenues et ses événements soient synchronisés les uns par rapport aux autres dans le temps, on construit ce qu'on appelle le graphe d'activités du programme (PAG). Le PAG donne une vue sur l'exécution du

programme, et constitue un point de départ pour effectuer l'analyse de performance du programme parallèle. Il est formé par des nœuds, qui représentent des événements qui se

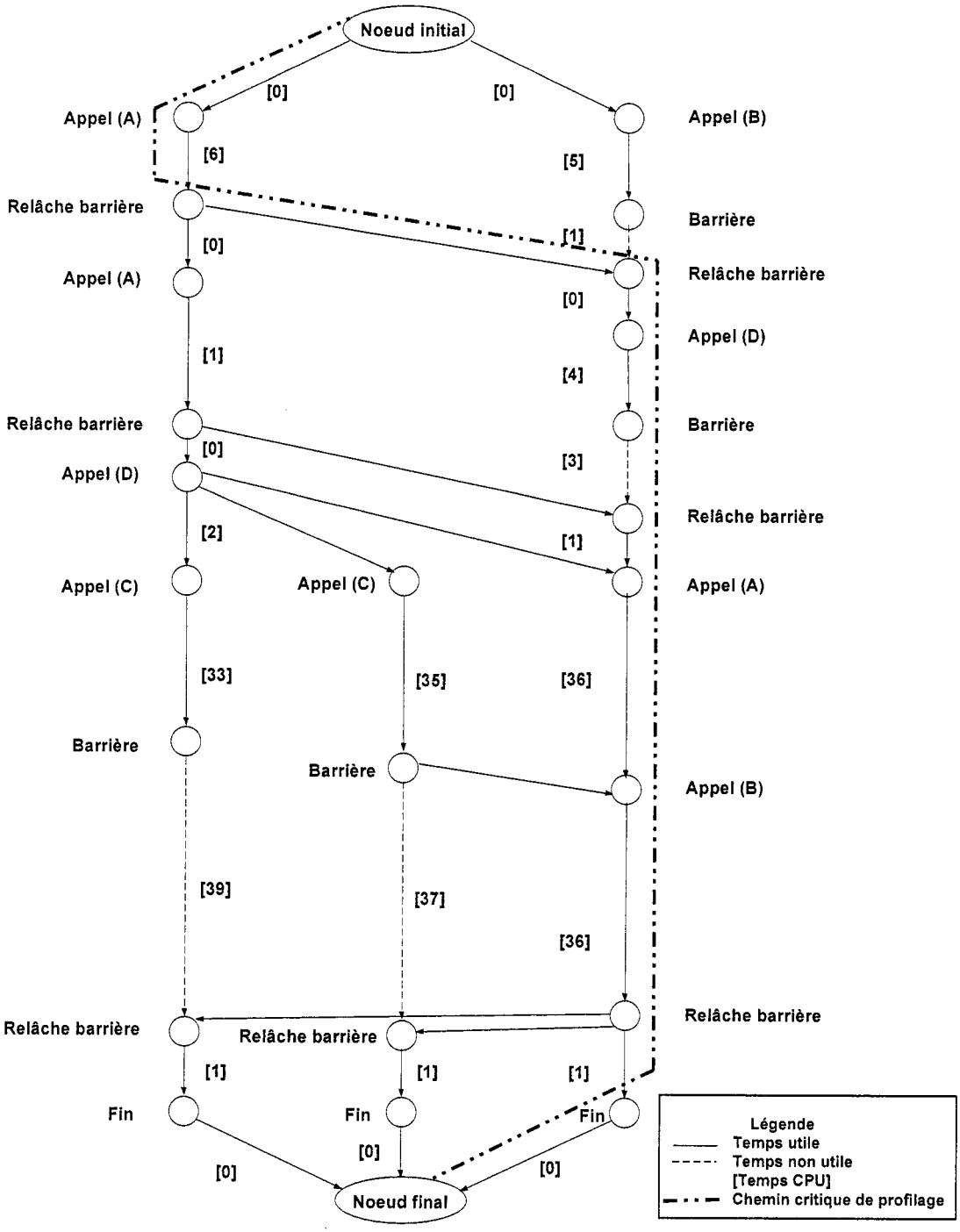

FIGURE 1.1 : Graphe d'activités d'un programme (PAG) parallèle sur 3 nœuds.

sont produits durant l'exécution du programme (appel de procédure, appel système ...) et des arcs qui représentent l'ordonnancement des événements au sein du même processus ou la synchronisation de la dépendance entre différents processus. Chaque arc est étiqueté par le temps consommé par le processeur

La figure 1.1 montre un simple PAG pour un programme parallèle avec 3 processus. Les lignes solides représentent le temps CPU utile et les lignes non continues représentent le temps CPU non utile, tel que l'attente de déblocage d'un verrou.

1.2 Métriques d'analyse de performance des programmes parallèles

1.2.1 Chemin critique de profilage

Le chemin critique de profilage [6] d'un programme parallèle est le chemin à travers le PAG ou le microprocesseur a passé le plus de temps. Le temps microprocesseur non productif, tel que l'attente de déblocage d'un verrou, se voit attribuer la valeur 0.

Le temps passé dans les procédures qui forment le chemin critique de profilage est la raison pour laquelle le programme a pris tout ce temps pour s'exécuter. Ainsi, le temps d'exécution du programme ne sera pas réduit tant que le temps passé dans l'une de ces procédures ne sera pas réduit.

Le PAG est un graphe orienté acyclique et aucun arc n'a un poids négatif. Ainsi, une variante de l'algorithme distribué du plus court chemin, décrit par Chandy et Misra [7], peut être utilisée pour déterminer le chemin critique de profilage. L'algorithme [8] commence depuis le nœud initial et traverse tous les nœuds du graphe en faisant passer des messages de chaque nœud aux nœuds suivant. Chaque message contient la valeur du plus long chemin jusqu'au nœud courant. Aux nœuds de divergence (nœuds avec un seul arc entrant et deux arcs sortant), le message est dupliqué et envoyé à travers chaque arc sortant. Aux nœuds de convergence (nœuds avec deux arcs entrant et un seul arc sortant), seulement le plus long chemin est pris en compte. La première phase de l'algorithme se termine lorsque le dernier nœud du graphe reçoit les messages de tous ses arcs entrants. Une fois le chemin déterminé, une seconde passe dans le sens inverse est effectuée à travers le graphe. Cette action traverse le chemin critique, pour calculer le temps consommé, par chaque procédure située sur le chemin critique.

1.2.2 Quartz NPT profiling

La métrique Quartz [9] est calculée en glissant une règle horizontalement à travers le PAG. A chaque nœud, on arrête la règle et le temps passé dans chaque procédure depuis le dernier nœud est calculé, on le note par exemple Tp. Puis, pour chaque processus i faisant un travail utile, on divise Tp par le nombre de processus faisant un travail utile et on ajoute ce résultat au cumulatif de la métrique Quartz, attribuée à la procédure

s'exécutant sur le processus en question. La figure 1.2 montre un exemple de calcul de cette métrique.

Pour mieux clarifier le principe de cette métrique, les deux premières passes de l'algorithme sont expliquées ici. Au premier arrêt, l'algorithme calcule le temps écoulé depuis le dernier nœud, jusqu'au nœud courant dans le processus 3, et qui est 33. Donc, Tp est égal à 33 pour les trois processus, et les trois processus étaient actifs dans cet intervalle. Pour le processus 1, la métrique de la procédure A sera égale à 33 / 3, pour le processus 2, la métrique de la procédure C sera égale à 33 / 3 et la même chose pour la procédure C dans le processus 3.

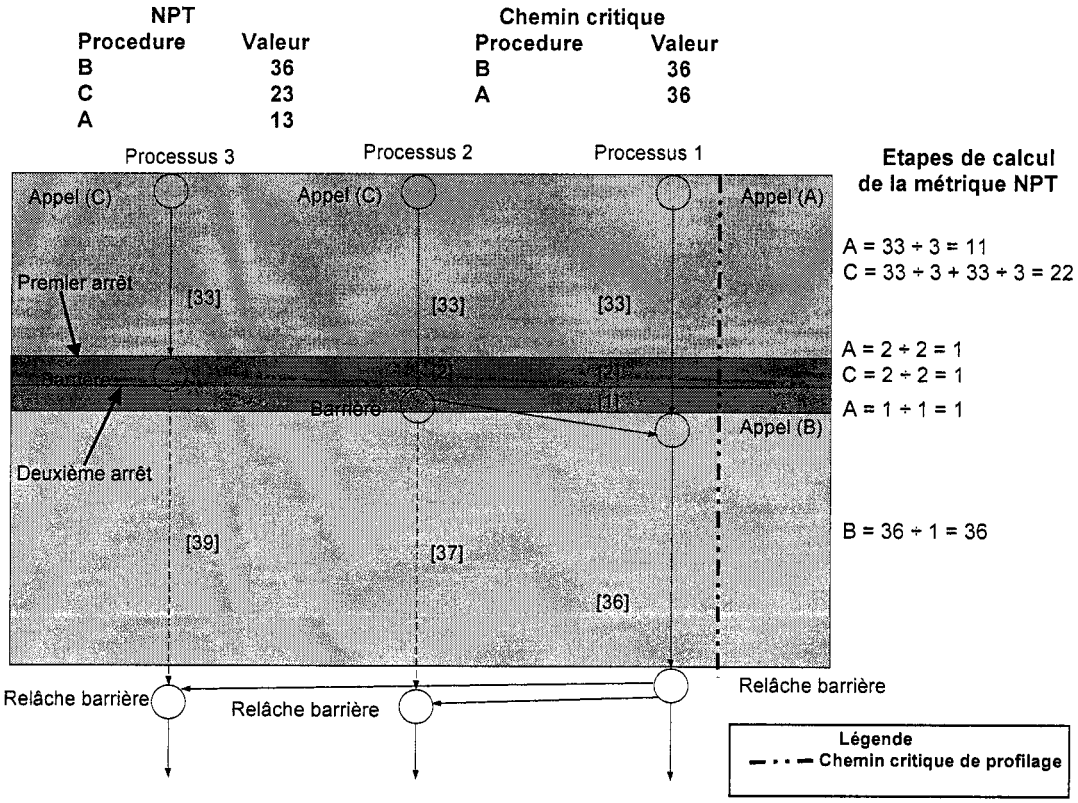

FIGURE 1.2 : Quartz NPT.

Au deuxième arrêt, l'algorithme calcule le temps écoulé depuis le dernier nœud jusqu'au nœud courant dans le processus 2, et qui est 2. Donc, Tp est égale à 2 pour les deux processus, et dans cet intervalle deux processus étaient actifs. Ainsi, pour le processus 1, la métrique de la procédure A sera égale à 2 / 2, et la métrique de la procédure C sera égale à 2 / 2 pour le processus 2. Et ainsi de suite, jusqu'au parcours total du PAG. Bien sûr, les métriques de chaque procédure sont accumulées à chaque arrêt de l'algorithme.

Avant de discuter d'autres métriques d'analyse de performance de programmes parallèles, une comparaison des deux techniques présentées jusqu'ici sera effectuée. D'après le PAG de la figure 1.2, la métrique Quartz calculée pour la procédure B est la plus grande. Ainsi, la métrique Quartz suggère que la procédure B doit être optimisée en premier lieu, afin de réduire le temps d'exécution total du programme. En effet, si on met le temps d'exécution de la procédure B à 0, le temps d'exécution total du programme sera réduit de 72 à 36. Par contre, la métrique Chemin Critique de Profilage de la procédure A est égale à celle de la procédure B. Ainsi, on peut optimiser soit la procédure A ou soit la procédure B pour réduire le temps d'exécution du programme, ce qui n'est que partiellement vrai. Par exemple, si on met le temps d'exécution de la procédure A à zéro, le temps d'exécution total du programme sera toujours 72, et on ne réalise aucun gain en performance.

1.2.3 Temps en réserve

La métrique du temps en réserve est basée sur le chemin critique. Elle détermine le gain en performance suite à une optimisation d'une procédure figurant sur le chemin critique. Ainsi, chaque procédure se voit attribuer une valeur du temps en réserve, qui indique jusqu'à quel point cette dernière peut être modifiée, avant que le chemin critique ne soit altéré. La métrique du temps en réserve est calculée en utilisant l'algorithme décrit dans [10].

L'idée de la métrique du temps en réserve est que le fait d'optimiser une procédure, figurant sur le chemin critique, n'implique pas que le temps d'exécution du programme sera réduit, car il se peut qu'il y ait un deuxième chemin presque critique, juste un peu moins long que le premier. Ainsi, la métrique du temps en réserve tient compte de ces chemins secondaires et de leurs relations avec le chemin critique.

On va calculer la métrique du temps en réserve et du chemin critique de profilage pour le PAG de la figure 1.3. Cet exemple montre que la métrique du chemin critique indique qu'il faut optimiser la procédure A. Ainsi, si on met le temps écoulé dans la procédure A égal à 0, cela réduira le temps d'exécution total du programme de 10 à 9. Par contre, la métrique du temps en réserve indique qu'il faut optimiser la procédure D. Si on met le temps écoulé dans la procédure D égal à 0, cela réduira le temps d'exécution total du programme de 10 à 7.

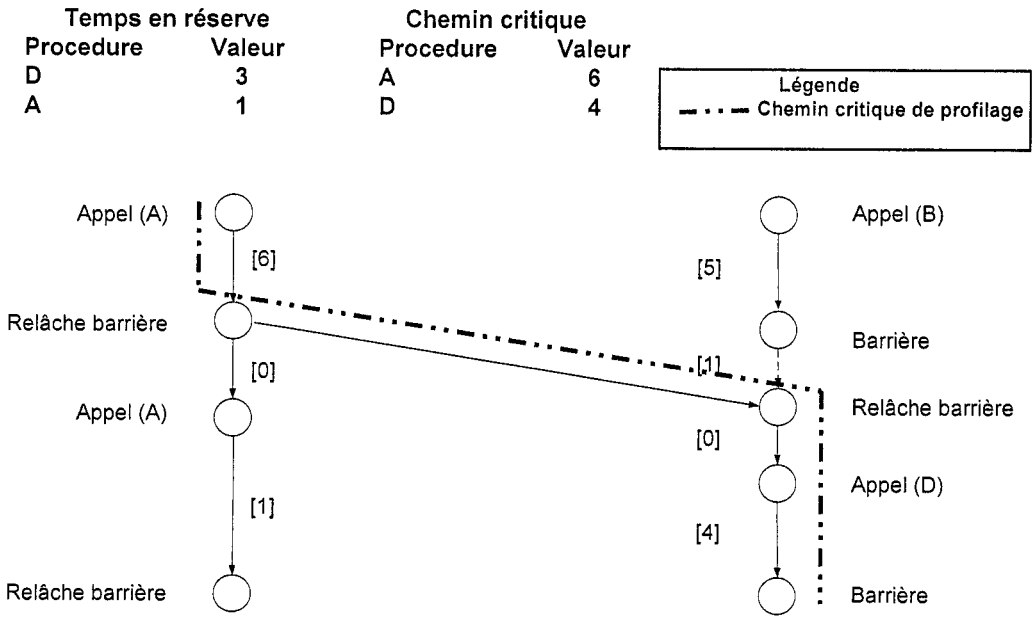

FIGURE 1.3 : Temps en réserve

1.2.4 Remise à zéro logique

L'idée de cette métrique est de sélectionner une procédure et de mettre ses arcs à zéro, puis de calculer la longueur du nouveau chemin critique. La différence entre la longueur du nouveau et de l'ancien chemin critique est l'estimation de l'amélioration, qu'on peut obtenir suite à une optimisation de la procédure sélectionnée.

Une alternative de cette métrique dans [6] consiste à éditer le code source de l'application, puis à enlever la procédure sélectionnée, et enfin à exécuter de nouveau le programme pour constater le gain en performance. La difficulté à utiliser cette technique, réside dans le fait d'enlever une procédure du code source sans affecter le reste du programme.

Pour conclure, il existe d'autres métriques ou des alternatives à celles mentionnées ci-dessus et il n'y a pas une métrique nécessairement plus performante qu'une autre. Tout dépend de la nature du programme parallèle traité. Ainsi, comme on a vu dans les figures 1.2 et 1.3, la métrique du chemin critique de profilage n'a pas bien déterminé la procédure à optimiser. Cependant, la métrique du chemin critique de profilage donne des bons résultats dans la plupart des cas.

Idéalement, l'outil d'analyse de performance parallèle doit proposer à l'utilisateur plusieurs techniques d'analyse du PAG. Ainsi, le programmeur pourra voir les points faibles de son programme sous différents angles. N'oublions pas, néanmoins, que le plus important élément dans l'analyse de performance c'est l'expérience et l'instinct du programmeur.

1.3 Système de temps à haute résolution

Ce système est basé sur le registre compteur de cycles d'horloge du microprocesseur, disponible dans la plupart des microprocesseurs récents des différentes architectures. Ce registre est incrémenté à chaque cycle d'horloge du microprocesseur. Donc, si on prend un microprocesseur cadencé à une fréquence d'horloge égale à 1 GHz, la résolution offerte par le registre compteur de cycles d'horloge serait égale à 1 nanoseconde. Bien sûr, de nos jours, les microprocesseurs avec une fréquence d'horloge à 1 GHz sont déjà dépassés.

LTT utilise actuellement ce registre pour estampiller les événements, puisqu'il peut tracer des événements successifs dont la différence de temps est quelques nanosecondes, et que le système de maintien du temps de Linux n'offre qu'une résolution de l'ordre de la microseconde. Notons que les versions actuelles du noyau de Linux utilisent ce registre, s'il est disponible sur l'architecture sous laquelle le système tourne, pour interpoler le temps avec précision. Cela ne change pas la résolution de l'horloge de Linux.

Un problème subsiste néanmoins. Le travail des horloges consiste à compter les cycles de l'oscillateur, qui est généralement basé sur un cristal de quartz, puis à effectuer les conversions nécessaires pour sauvegarder les résultats dans le registre compteur.

Il se trouve que les horloges [4] sont caractérisées par un taux de dérive. Ce dernier est le décalage entre une horloge et une horloge de référence (horloge presque parfaite qui maintient le temps réel), par unité de temps mesurée par l'horloge de référence. Pour une horloge basée sur un cristal en quartz ce taux est de l'ordre de 10^{-6} secondes/seconde, donc une déviation d'une seconde à chaque 11,6 jours. Le décalage d'une horloge est la différence entre l'horloge et celle de référence à un moment donné. Ce décalage peut changer dans le temps en raison de la dérive.

Enfin, la variation de la température fait varier la fréquence de l'oscillateur au quartz, ce qui a un impact direct sur la dérive de l'horloge, et par ricochet sur le décalage. Bien que des techniques de conception soient utilisées pour contrer ce problème, elles ne peuvent l'annuler.

En résumé, tous ces facteurs résultent en une imprécision de l'horloge et causent un décalage de l'horloge qui varie au cours du temps, peu importe l'exactitude avec laquelle l'horloge a été synchronisée initialement par rapport à une horloge de référence.
Dans [21], une étude a été faite sur la dérive et la stabilité d'une horloge basée sur le registre compteur de cycles d'horloge. Dans cette expérience, la fréquence du microprocesseur est estimée au départ en se basant sur l'oscillateur standard de l'ordinateur. Cette estimation permet de convertir le nombre de cycles d'horloge écoulés en temps. Dans l'étude, on mentionne que cette manière de faire implique que l'horloge obtenue dérive selon l'erreur de l'oscillateur standard de l'ordinateur.
A l'encontre de cette étude, on se propose d'étudier le registre compteur de cycles d'horloge sans l'intervention d'autres composantes et d'effectuer des expériences sur une série de processeurs de différentes architectures et constructeurs afin de valider les résultats obtenus.

1.3.1 Temps sous Linux

Linux utilise deux sources matérielles pour maintenir son horloge à jour: l'horloge temps réel (RTC) et le temporisateur à intervalle programmable (PIT).
Le RTC, est un circuit disponible dans tous les ordinateurs. Son rôle est de maintenir le temps même lorsque l'ordinateur est éteint. Il est représenté par le composant 146818 de Motorola dans les ordinateurs compatibles IBM. La résolution du RTC est limitée, de l'ordre de la seconde, et sa dérive d'horloge est grande. C'est ainsi que Linux ne l'utilise

qu'au démarrage de l'ordinateur pour déterminer la date et l'heure actuelle maintenue par ce dernier. Par la suite, pour garder la date et l'heure à jour, Linux se fie au PIT qui est représenté par le circuit 8254 d'Intel dans les ordinateurs compatibles IBM. Ce dernier est alimenté par la fréquence de sortie d'un oscillateur au quartz égale à 14,31818 MHz divisée par 12, ce qui est équivaut à 1,1193182 MHz. Ainsi, la résolution du PIT est égale à 0,838 microseconde.

Au démarrage de l'ordinateur, Linux programme le PIT, de sorte qu'il génère une interruption périodiquement, la valeur de la constante HZ sert à déterminer cette fréquence et elle est égale à 100 Hz pour la plupart des ordinateurs compatibles IBM, une interruption à chaque 10 ms. C'est ainsi qu'à chaque interruption générée par le PIT, Linux va mettre à jour son horloge. D'une manière simplifiée, il va ajouter 10 ms à son horloge actuelle. Notons que Linux utilise le registre compteur de cycles d'horloge, dans le cas où ce dernier est disponible dans l'ordinateur où il tourne, pour interpoler le temps.

1.3.2 Registre compteur de cycles horloge du microprocesseur

La majorité des microprocesseurs récents disposent du registre compteur de cycles d'horloge, qui est incrémenté à chaque cycle du microprocesseur. La mise à jour du registre se fait par le matériel et sa lecture ne prend que quelques cycles d'horloge du microprocesseur. En plus, la fréquence des microprocesseurs de nos jours dépasse 1

GHz, et donc dépasse la résolution cherchée de 1 nanoseconde. Certains détails diffèrent, selon l'architecture du microprocesseur. Ainsi, il se peut que le registre soit :

- d'une taille de 32 ou 64 bits;
- lisible en mode utilisateur ou non;
- en lecture seule ou en lecture et écriture.

Par exemple, pour les processeurs x86, ce registre est nommé TSC (TimeStamp Counter). C'est un registre 64 bits et il a été introduit à partir des processeurs Pentium 1. Pour les processeurs Alpha, il est représenté par le registre PCC (Processor Cycle Counter).

Les processeurs étudiés sont ceux des plates-formes les plus répandues : i386 (de différents constructeurs Intel, AMD et VIA) et Itanium. Notons que d'autres architectures peuvent être étudiées, selon les contraintes de temps et de disponibilité de matériel.

Pour l'étude de ce registre, il faut une source de temps externe précise, qui peut nous délivrer un signal périodiquement afin d'observer le registre compteur de cycles d'horloge. Dans [23] on discute de la précision de ce signal qui dépend entre autres, du délai en dehors de l'ordinateur et à l'intérieur de l'ordinateur. Le délai en dehors de l'ordinateur (ex : longueur du câble : 1µs / 200) est constant et peut être compensé. Par contre, le délai à l'interne de l'ordinateur est variable et il correspond à la différence entre le moment où le signal arrive sur l'ordinateur et le moment où il est pris en compte par le module sériel. Dans la même étude [23], on montre que pour un Pentium III cadencé à 860 MHz, le délai moyen est 8.31 microsecondes avec un écart-type de 0.36.

Ce délai s'explique majoritairement par la latence du système d'exploitation, Linux étant un système multi-utilisateurs et multitâches. N'oublions pas la latence du matériel qui s'occupe de la gestion des interruptions, même si cette dernière est moins variable.

Pour notre étude, l'interception du signal est faite au plus bas niveau possible (dès que Linux intercepte la demande d'interruption), afin de réduire au maximum le délai à l'interne de l'ordinateur.

1.4 Synchronisation des horloges dans un système réparti

La nouvelle version de LTT doit être utilisée dans des grappes de calcul scientifiques. Des traces d'exécution des programmes parallèles sur plusieurs nœuds seront générées et corrélées, afin de savoir, lorsqu'un nœud faisait un certain traitement, ce que faisaient les autres nœuds. Cela implique la nécessité d'avoir un bon degré de synchronisation des horloges de la grappe.

Le problème de synchronisation des horloges dans un système réparti n'est pas nouveau. Les horloges d'un système réparti ne comptent pas le temps avec la même fréquence et ne sont jamais parfaitement synchronisées au départ, et ainsi elles divergent au cours du temps.

Le type et le degré de synchronisation désiré influencent la méthode de synchronisation des horloges à utiliser. Deux types de synchronisation existent : Synchronisation interne,

qui consiste à synchroniser les horloges du système en question par rapport à une horloge du système et l'utiliser comme horloge de référence, sans se soucier de la différence qu'on peut avoir avec une horloge de référence réelle. Synchronisation externe, qui consiste à utiliser une source externe pour obtenir le temps réel, comme l'utilisation d'un récepteur GPS [19] (système de positionnement global) qui est formé par 24 satellites mis en orbite. Ces derniers sont munis d'horloges atomiques pour maintenir le temps qu'ils envoient dans les signaux interceptés par les récepteurs GPS. Le récepteur GPS sera branché à un serveur de temps du système réparti, qui sera utilisée pour synchroniser le reste des horloges.

Les horloges les plus précises dans le monde sont à base d'oscillateurs atomiques, dont l'erreur de précision [4] est très faible (10^{-13} secondes/seconde). Le temps de référence peut être le temps UTC. Le temps universel coordonné UTC est un standard international pour le temps. Il est basé sur des horloges atomiques, pour préserver l'exactitude du temps. Les signaux UTC sont synchronisés et diffusés par des stations radios ou par des satellites. Ainsi, le temps UTC peut être obtenu à partir du système de positionnement global GPS. Pour recevoir le temps UTC, il suffit de disposer d'un récepteur GPS.

Le problème qui s'ajoute à celui des horloges est le temps de transmission réseau. En effet un nœud qui désire synchroniser son horloge par rapport à celle du serveur de temps a besoin d'échanger des messages avec le serveur à travers le réseau qui les relie. Malheureusement, le temps de transmission exact d'un message sur un réseau n'est pas connu puisqu'il est sujet à des variations suivant l'état de la charge du réseau. Les seuls

paramètres connus sont le temps minimal de transmission et le temps maximal (dans le cas d'un système réparti synchrone). Dans le cas d'un système réparti asynchrone, le temps maximal est supérieur au temps minimal mais inconnu. La plupart des systèmes répartis sont asynchrones.

Les sections suivantes analysent les solutions qui ont été proposées et qui sont utilisées pour remédier au problème de synchronisation des horloges dans un système réparti.

1.4.1 Méthode de Cristian

Cristian[4] a suggéré l'utilisation d'un serveur de temps, connecté à un équipement qui reçoit le signal d'une source de temps UTC. Le serveur, sur requête, envoie son horloge dans son message de réponse.

Cristian a observé que le temps d'aller et de retour d'un message dans un système asynchrone est raisonnablement petit. Ainsi, il a décrit un algorithme basé sur un principe probabiliste. Sa technique effectue la synchronisation si et seulement si le temps d'aller et de retour entre le client et le serveur est suffisamment petit par rapport à la précision demandée.

Le client demande le temps au serveur, puis reçoit le temps dans le message de réponse du serveur. Le client enregistre le temps d'aller et de retour $T_{aller\text{-}retour}$, qui est le temps écoulé à partir de l'envoi de son message, jusqu'à réception de la réponse du serveur. Dans le cas idéal, on assume que le temps que le message prend dans l'aller est égal à

celui du retour, et le client peut régler son horloge à $t + \frac{T_{aller\_retour}}{2}$. Dans le cas réel, le temps d'aller est différent du temps de retour. C'est ainsi, que l'horloge du client serait ajustée à $t + \frac{T_{aller\_retour}}{2}$ avec une erreur de précision égale à $\frac{T_{aller\_retour}}{2} - \min$ (min est le temps minimal de transmission).

Si on a des $T_{aller\text{-}retour}$ qui sont plus grands que la moyenne, ils seront ignorés, car l'erreur de précision serait grande. Pour y remédier, plusieurs requêtes peuvent être faites par le client dans un espace de temps différé (si jamais il y a congestion dans le réseau, elle sera évitée de cette manière), afin d'obtenir des $T_{aller\text{-}retour}$ petits et par conséquent une bonne précision de l'estimation du temps par le client.

1.4.2 L'algorithme de Berkley

Gusella et Zatti [11] ont décrit un algorithme pour une synchronisation interne, qu'ils ont utilisé pour leur ensemble d'ordinateurs qui tournent avec Berkley UNIX. Leur idée est de désigner un ordinateur comme maître. Ce dernier fait des requêtes périodiquement aux autres ordinateurs, appelés esclaves, dont la synchronisation de l'horloge est demandée. Ces esclaves répliquent au maître avec un message contenant leur heure actuelle. Similairement à la technique de Cristian, le maître estime l'heure de l'esclave interrogé. Le maître élimine toute lecture dont le temps d'allée et de retour est supérieur à celui du temps nominal maximal d'aller et de retour entre le client et le maître et dont dépend l'exactitude de l'algorithme.

A l'opposé de la technique de Cristian, ici le maître n'envoit pas aux esclaves le temps auquel ils doivent s'ajuster, mais plutôt la valeur de décalage avec laquelle ils doivent ajuster leur horloge. Cela évite l'incertitude causée par le délai de transmission réseau. Cette valeur peut être positive ou négative.

1.4.3 Optimisation de la synchronisation

D'après les deux techniques mentionnées ci-dessus, on note que la synchronisation des horloges dans un système réparti dépend principalement de deux facteurs :

1. Le délai de transmission sur le réseau, et sa variabilité, comme on l'a vu plus haut dans la technique de Cristian.
2. La couche du système d'exploitation dans laquelle est effectuée la lecture du temps des programmes client et serveur, qui interagissent entre eux pour une synchronisation. Ce qui sera étudié à la section 4.1.3

Le premier facteur dépend de la technologie et de l'architecture du réseau utilisé. Celles-ci déterminent la latence de transmission d'un message, ainsi que la vitesse de transmission dans le réseau. La formule suivante [12] illustre le temps de transmission simple d'un message dans un réseau :

$$t_c(n) = t_0 + n \frac{1}{b_{eff}}$$

t_0 : est la latence du réseau

$n \frac{1}{b_{eff}}$: n est la taille du message à transmettre. b_{eff} est la bande passante du réseau.

Dans le cas d'un réseau avec plusieurs nœuds à travers lesquels le message doit transiter, la formule devient :

$$t_c(n,h) = \quad t_0 + \quad ht_1 + \quad n\frac{1}{b_{eff}}$$

h t_1 : h est le nombre de nœuds intermédiaires que doit traverser le message et t_1 est le délai associé à un nœud.

Dans le cas d'un réseau avec plusieurs nœuds qui sont en compétition pour l'accès au médium de transport, la formule devient :

$$t_c(n,s) = \quad t_0 + \quad n\frac{s}{b_{eff}}$$

s : est le nombre de nœuds en compétition.

Pour avoir un bon degré de synchronisation par rapport à la précision demandée, il faudra que la bande passante du réseau soit la plus grande possible, que la structure réseau du système réparti soit la plus simple possible, et que la latence du réseau soit la plus petite possible ainsi que sa variabilité. Le dernier point est le plus important, car la variabilité de la latence a un impact direct sur le délai de transmission réseau. Les solutions proposées pour la synchronisation des horloges dans un système réparti supposent que le temps d'aller et de retour entre le client et le serveur sont similaires. Ce n'est pas réellement le cas, car il y a toujours une différence entre les deux temps, et c'est cette différence qui caractérise majoritairement l'erreur de précision de la synchronisation des horloges.

Le tableau qui suit donne les caractéristiques [12] de différents types d'architectures réseaux.

TABLEAU 1.1 : Spécifications de différentes architectures réseaux.

	Fast Ethernet	Gigabit Ethernet	SCI	ATM	Myrinet
Structure réseau	bus	bus	commuté	commuté	Commuté
Latence dans un seul sens	20 µs	20 µs	5 µs	120 µs	5 µs
Bande passante	100 Mbps	1 Gbps	4 Gbps	155 Mbps	1.2 Gbps
Longueur du câble par lien	200 m	200 m	10 m	100 m	10 m

Maintenant, il est temps d'expliquer le deuxième facteur. En fait, si la lecture du temps des horloges, dans les programmes client et le serveur est implantée au niveau *application* du model réseau de l'organisation internationale de standardisation (ISO), la latence du système d'exploitation vient s'ajouter au temps de transmission entre le client et le serveur. Cette latence est sujette à des variations, suivant la maturité du système d'exploitation et la charge actuelle du système. Par contre, si elle est implémentée à un niveau plus bas, cette latence diminue. Dans [13], une solution de synchronisation basée sur la technique de Cristian avec quelques modifications a produit de bons résultats. Elle utilise les processeurs des cartes d'interface réseau pour estampiller le temps, afin d'éliminer le bruit du système sous-jacent, puisque le microprocesseur de l'ordinateur ne sera pas utilisé pour estampiller le temps. Plus précisément, un nœud est désigné comme serveur, ce dernier synchronise les horloges des autres nœuds à tour de rôle, comme illustré dans la figure 1.4. Pour ce faire, il calcule la différence de temps entre lui et un

nœud à synchroniser, puis il renvoie cette différence de temps au nœud en question, pour que ce dernier synchronise son horloge par rapport au serveur. La différence de temps est calculée comme suit :

$$T_{diff} = T'_0 - T'_1 - T_{net}$$

Avec

$$T_{net} = \frac{(T'_0 - T_0) - (T'_1 - T_1)}{2}$$

T_0 : Temps au noeud 0 au moment de l'envoi du message "MyriTime"

T_1 : Temps au noeud 1 au moment de la réception du message

T'_1 : Temps au noeud 1 au moment du renvoi du message

T'_0 : Temps au noeud 0 au moment de la réception du message renvoyé

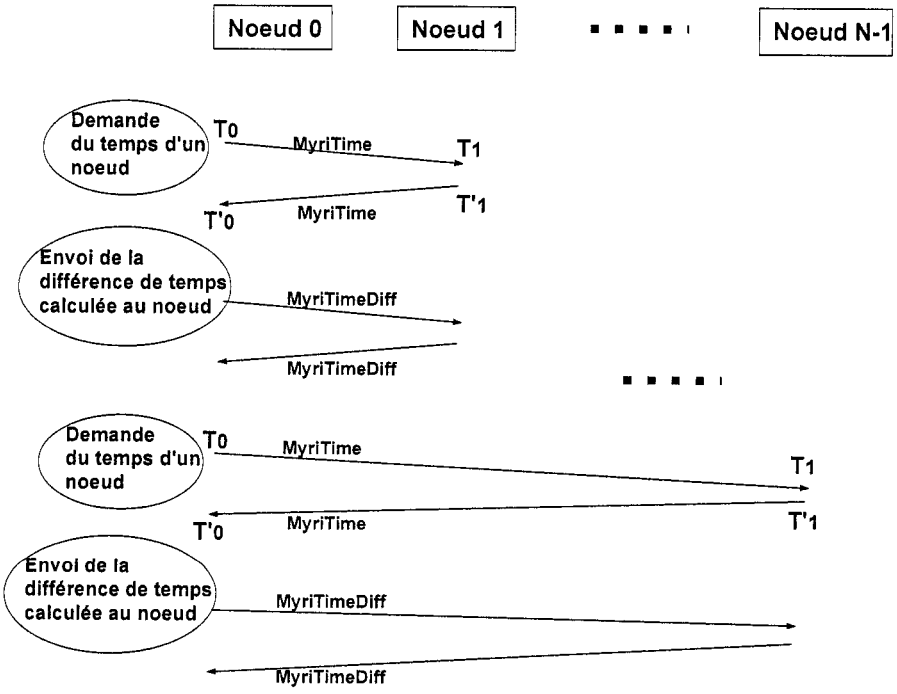

FIGURE 1.4 : Principe global de synchronisation

Le tableau 1.2 représente les résultats obtenus avec deux variantes de l'algorithme de synchronisation « adaptative et non adaptative ». La différence entre ces deux algorithmes réside dans le fait que la synchronisation adaptative ajuste la durée de la re-synchronisation des nœuds selon l'état actuel du système. Cette technique, bien qu'elle reste une solution à usage non général, dépasse la synchronisation atteinte par NTP, technique plus largement applicable. Dans un réseau local à 100 Mbps, connecté à un serveur de temps primaire, la synchronisation qu'on peut atteindre avec cette technique est de l'ordre de la microseconde.

TABLEAU 1.2 : Erreur d'horloge et déviation standard de l'algorithme adaptatif et non adaptatif de synchronisation avec système en repos puis en charge.

	Système repos. Erreur (µs)	Système repos. Écart type (µs)	Système chargé. Erreur (µs)	Système chargé. Écart type (µs)
Non adaptatif	0.80	1.00	2.54	2.83
Adaptatif	0.65	0.95	1.26	1.45

Malheureusement, la majorité des cartes d'interface réseau ne disposent pas de processeur propre. Il est donc proposé que la lecture du temps du côté client et serveur se fasse au plus bas niveau, plus précisément dans le code source du programme de gestion de la carte d'interface réseau. Ainsi, le bruit causé par le système d'exploitation est minimisé. L'architecture réseau la plus répandue est Ethernet, vu son rapport qualité/prix. Le problème de cette architecture est sa grande latence qui peut augmenter la variabilité du délai de transmission.

Pour mieux quantifier l'erreur de précision, qui peut être engendrée par la latence dans un réseau Ethernet, une étude de la symétrie du délai de transmission de message entre

le client et le serveur sera faite. Dans le cas de l'algorithme de Cristian, lorsque le client reçoit le message de réponse avec le temps du serveur, il ajuste son horloge à celle du serveur plus le temps aller et retour du message divisé par deux. Donc, il suppose que le temps d'aller est égal au temps de retour, ce qui n'est pas parfaitement le cas.

Ensuite le mécanisme de diffusion générale sera étudié. En principe, lorsqu'un nœud diffuse un message dans le réseau, le message peut être destiné à tous les nœuds du réseau. Dans un réseau Ethernet, on sait que chaque nœud qui y est relié est à l'écoute du médium de communication, pour savoir si le message diffusé lui est destiné. Ainsi, on peut supposer que le message atterrit sur la carte réseau de chaque nœud pratiquement au même moment. Un message peut donc être envoyé en mode diffusion générale, au cours de l'exécution d'un programme parallèle, et chaque nœud qui le reçoit l'estampille avec son horloge locale.

Puisque le message parviendra à tous les nœuds au même moment, ce dernier sera utilisé comme un repère de temps pour synchroniser les horloges des nœuds. Plus précisément, au moment de la corrélation des traces, un nœud sera utilisé comme référence de temps, avec lequel les autres nœuds vont se synchroniser, en calculant la différence de leurs horloges estampillées, lorsqu'ils ont reçu le message. Le diagramme de séquence dans la figure 1.5 illustre bien cette idée :

Le nœud de référence est le nœud 1. Ainsi l'horloge du nœud 2 doit être ajustée avec (t1a-t2a), et ainsi de suite pour les autres nœuds du système.

Pour conclure ce chapitre, il est suggéré que la version parallèle de LTT implémente au moins deux métriques d'analyse de performance, et que la métrique chemin critique de profilage soit incluse par défaut, car elle donne de bons résultats la plupart du temps.

Une fois que les expériences sur le registre compteur de cycles d'horloge auront été effectuées, la possibilité d'utiliser ce registre comme système de temps pour LTT pourra être examinée. Finalement, les tests réseaux vont permettre de conclure sur la précision de synchronisation des horloges des nœuds d'une grappe de calcul scientifique qui peut être atteinte.

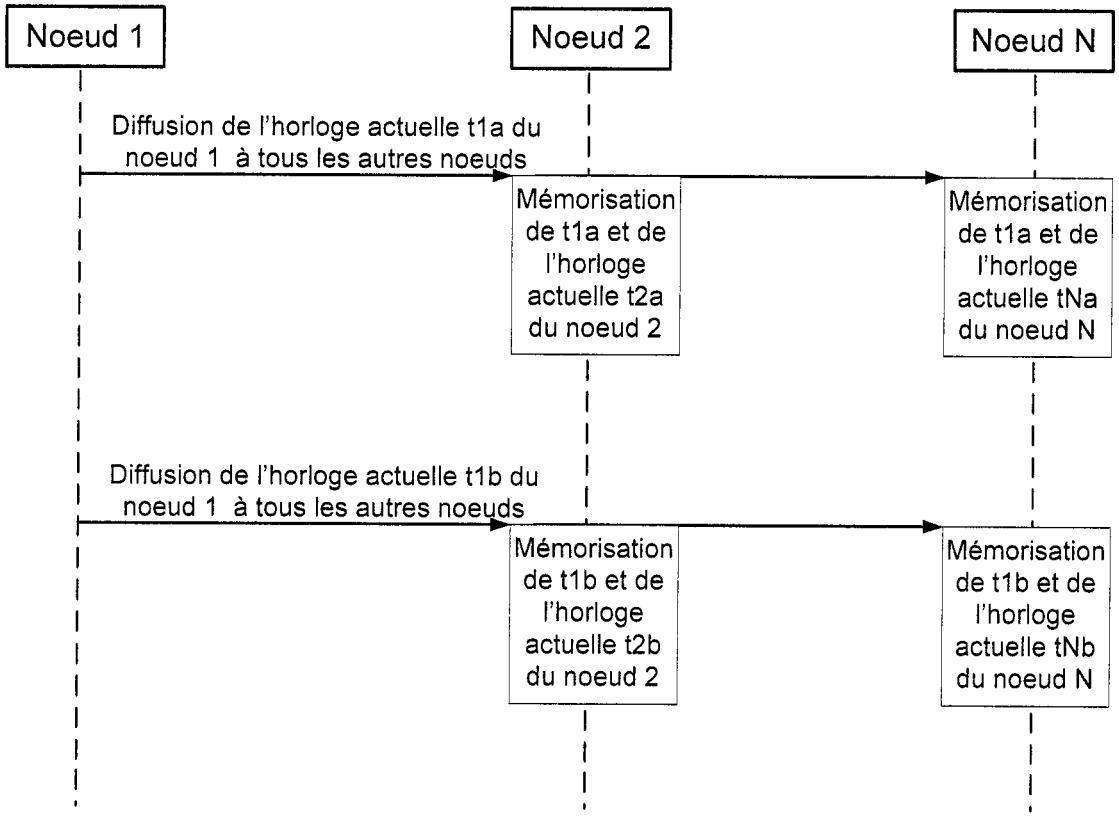

FIGURE 1.5 : Diagramme de séquence pour la synchronisation des horloges des nœuds en utilisant la technique de diffusion général

Chapitre 2

Concepts de base

2.1 Gestion des interruptions sous LINUX

2.1.1 Introduction

Le système d'exploitation permet l'utilisation des composantes de l'ordinateur (clavier, carte réseau...) d'une manière facile et compréhensible à l'être humain. Ainsi, le système d'exploitation doit interagir avec une multitude de matériel. Pour y parvenir, deux techniques existent : interruption et requêtes répétitives. La deuxième technique est peu utilisée de nos jours car elle consomme beaucoup de temps microprocesseur, car ce dernier doit continuellement demander au périphérique s'il a des informations prêtes à transmettre. La première technique est donc la plus répandue. Son principe est simple: lorsqu'un périphérique a besoin d'être servi, il fait une demande d'interruption au

microprocesseur, ce qui conduit ce dernier à laisser de côté ce qu'il est en train de faire, et servir le périphérique en question.

Le mécanisme de gestion des interruptions, utilisé sous Linux, permet un traitement simplifié des interruptions adapté aux différentes architectures. Ce modèle est constitué de deux parties. Une partie qui se charge de gérer les différences du matériel du système d'interruption relatif à chaque architecture et une partie qui se charge d'offrir une interface uniforme à l'utilisateur pour utiliser les interruptions. Par exemple, si on veut réserver une ligne de demande d'interruption pour un périphérique, la fonction système request_irq() se charge de le faire, peu importe l'architecture sur laquelle on travaille.

Notons bien qu'il y a deux catégories d'interruptions. La première catégorie *interruptions synchrones* et sont générées tandis que l'unité de contrôle du microprocesseur exécute une instruction. Elles sont appelées synchrones car elles ne sont émises qu'après que l'unité de contrôle termine l'exécution de l'instruction en cours. La compagnie Intel utilise son propre jargon et nomme ce type d'interruption Exception. Un exemple est celui d'un programme qui, en cours d'exécution, effectue une division par zéro et va provoquer l'exécution du gestionnaire d'exception *divide_error()*, pour signaler au programme en cours l'anomalie. Deuxième catégorie, *interruptions asynchrones* qui sont générées arbitrairement par les périphériques indépendamment de l'horloge du microprocesseur. Intel, nomme cette catégorie *Interruption*. Dans cette dernière catégorie, on distingue deux types d'interruptions. Les interruptions *masquables* et interruptions *non masquables*. Le premier type d'interruptions peut être masqué, ainsi les interruptions seront ignorées par le processeur tant qu'elles seront masquées. Ce

genre d'interruptions est généré par les périphériques d'entrée / sortie (disque dur, carte réseau...). Par contre, le deuxième type d'interruptions ne peut être ignoré par le microprocesseur, elles sont traitées aussitôt qu'elles lui parviennent. Ce genre d'interruptions provient de quelques événements spéciaux comme une panne du matériel.

D'une manière abstraite, comme on peut le voir dans la figure 2-1, les périphériques sont connectés à un contrôleur d'interruption programmable à travers des lignes de demande d'interruption (Irq0, Irq1...Irqn). Ce dernier a la tâche d'acheminer les requêtes d'interruptions au microprocesseur. Une fois que le microprocesseur accepte la demande d'interruption, il va chercher le code du gestionnaire d'interruption pour servir la demande.

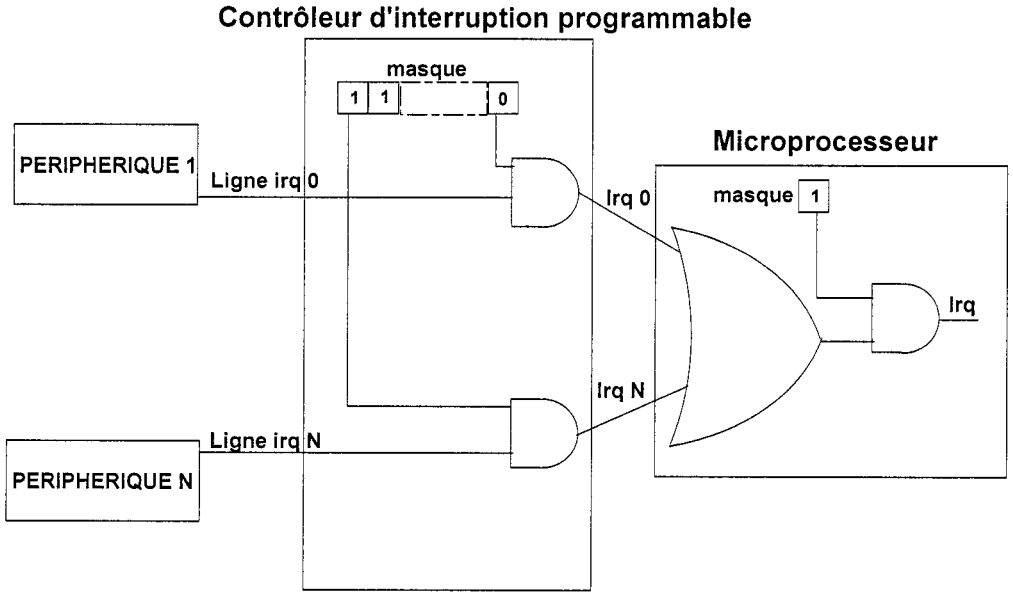

FIGURE 2.1 : Schéma global des interruptions

Il existe deux niveaux pour masquer les interruptions. Le premier, qui est au niveau du contrôleur d'interruption, permet de désactiver une ligne de demande d'interruption

spécifique. Les interruptions désactivées ne sont pas perdues car elles seront acheminées au microprocesseur aussitôt que la ligne IRQ correspondante aura été réactivée. Linux[14] utilise une structure abstraite 'hw_interrupt_type' déclarée dans le fichier d'en-tête usr/src/linux/include/irq.h pour gérer différents contrôleurs d'interruptions programmables. Les champs de la structure pointent vers les fonctions de gestion appropriées du contrôleur d'interruption programmable utilisé. Par exemple, la fonction *enable(unsigned int irq),* permet d'activer une ligne IRQ et la fonction *disable(unsigned int IRQ)* permet de désactiver une ligne IRQ.

```
struct hw_interrupt_type {
       const char * typename;
       unsigned int (*startup)(unsigned int irq);
       void (*shutdown)(unsigned int irq);
       void (*enable)(unsigned int irq);
       void (*disable)(unsigned int irq);
       void (*ack)(unsigned int irq);
       void (*end)(unsigned int irq);
       void (*set_affinity)(unsigned int irq, unsigned long mask);
};
```

Le deuxième niveau de masque, qui est au niveau du microprocesseur, permet une désactivation globale des interruptions. Ainsi, les interruptions masquables seront ignorées temporairement par le microprocesseur. Linux[15] met à notre disposition la fonction *cli()* pour désactiver les interruptions, et la fonction *sti()* pour l'activation des interruptions.

Chaque interruption est identifiée par un numéro variant de 0 à 255. Intel nomme vecteur ce chiffre de 8 bits non signés. Les vecteurs d'interruptions non masquables sont fixes, contrairement à ceux des interruptions masquables qui peuvent être changés en programmant le contrôleur d'interruption.

2.1.2 Contrôleur d'Interruption Programmable (PIC)

Le contrôleur d'interruption programmable joue un rôle d'intermédiaire entre les périphériques et le microprocesseur. D'un côté, ses broches d'entrées qui sont nommées *lignes de demande d'interruption* «IRQ», sont reliés aux lignes de sortie des périphériques, que ces derniers utilisent pour envoyer des requêtes d'interruption. De l'autre côté, il est connecté aux broches d'interruptions du microprocesseur. L'algorithme qui suit illustre le fonctionnement du PIC :

1. Superviser les lignes IRQ, pour vérifier si un périphérique a fait une demande d'interruption;
2. Si un signal d'interruption a eu lieu sur une ligne IRQ aller à 3, sinon retour à 1;
3. Convertir le signal reçu en un vecteur correspondant;
4. Enregistrer le vecteur dans un port d'entrée/sortie du PIC, pour permettre au microprocesseur de le lire via le bus des données;
5. Envoyer un signal au microprocesseur à travers son pin INTR, pour que ce dernier traite la demande d'interruption;
6. Attendre jusqu'à ce que le microprocesseur confirme qu'il a reçu le signal d'interruption en écrivant sur l'un des ports d'entrée/sortie du PIC. Puis, remettre à zéro la ligne INTR;
7. Revenir à l'étape 1.

Les lignes de demande d'interruption sont numérotées de 0 à 255. Ainsi, la première ligne d'interruption est notée IRQ0. Intel associe par défaut la ligne IRQn au vecteur n+32, donc la ligne de demande d'interruption 4 du port série se voit attribuée le vecteur 36. Cette correspondance peut être modifiée en programmant le PIC.

Le contrôleur d'interruption programmable varie selon l'architecture. Ici, on parlera des contrôleurs d'interruptions programmables utilisés dans l'architecture Intel 32 et Itanium, car elles feront l'objet de notre étude par la suite.

Architecture Intel 32

Système monoprocesseur

Intel a mis en place le circuit 8259A, qui joue le rôle du contrôleur d'interruption programmable. Ce dernier dispose de huit lignes de demande d'interruptions. Deux circuits 8259A seront connectés en cascade pour qu'on ait à notre disposition 15 lignes IRQ. La ligne de sortie INT du PIC esclave est connectée à la ligne IRQ 2 du PIC maître.

Système multiprocesseur

Dans ce genre de système, les choses se compliquent car plusieurs processeurs entrent en jeu, et le contrôleur d'interruption programmable devra être capable de délivrer une demande d'interruption à un processeur spécifique, ou à un groupe de processeurs. Le circuit 8259A n'a pas été prévu pour ce genre d'architecture.

Pour répondre aux besoins de l'architecture SMP (multiprocesseur symétrique), Intel a introduit un nouveau contrôleur d'interruption programmable avancé [16] et qui est implémenté par le circuit 16850. C'est le I/O APIC (Input/Output Advanced Programmable Interrupt Controller). Ce dernier est formé par 24 lignes de demande d'interruptions, une table de redirection d'interruptions formée de 24 entrées et une unité de message pour envoyer et recevoir les messages à travers le bus APIC. En plus, les récents microprocesseurs d'Intel incluent un contrôleur d'interruption programmable

local avancé (local APIC). Ce dernier dispose d'un registre 32 bits, d'une horloge interne, d'un temporisateur local et de deux lignes IRQ additionnelles. Tous les APIC locaux des microprocesseurs sont connectés aux I/O APIC via le bus APIC et les périphériques sont connectés aux I/O APIC, comme le montre la figure 2-2.

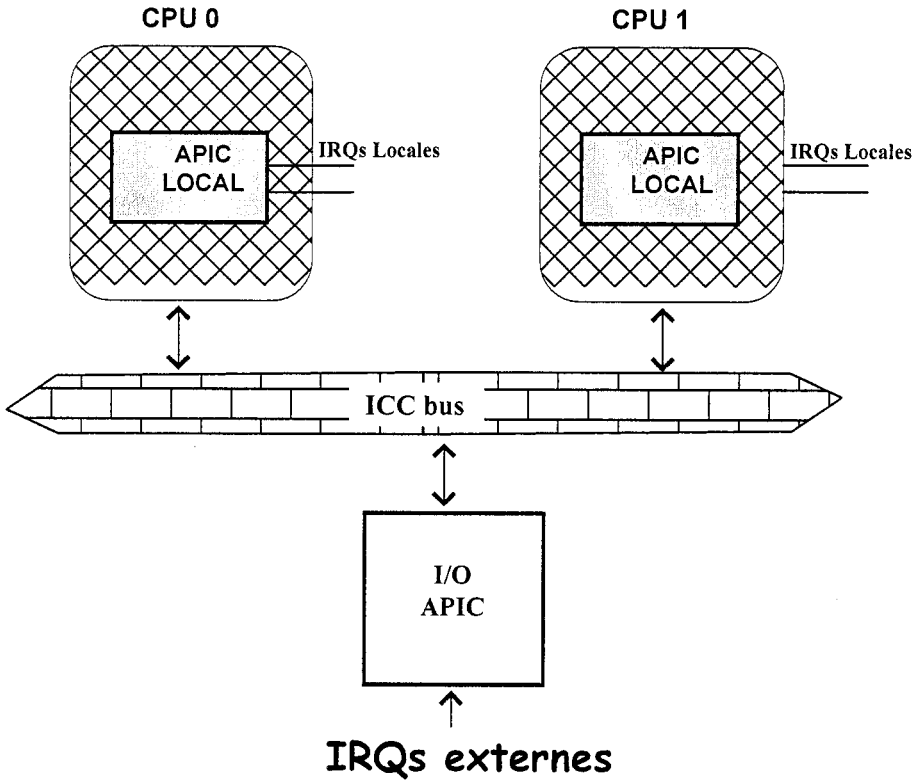

FIGURE 2.2 : Multi-APIC System

Dans Linux, la structure *IO_APIC_route_entry*, dans le fichier d'en-tête usr/src/linux/include/linux/asm-i386/io_apic.h, représente différents registres du IO_APIC qu'on peut programmer.

```
struct IO_APIC_route_entry {
      __u32 vector      :  8,
            delivery_mode      :  3, /* 000: FIXED
                                  * 001: lowest prio
                                  * 111: ExtINT
                                  */
            dest_mode    :  1, /* 0: physical, 1: logical */
            delivery_status    :  1,
            polarity     :  1,
            irr          :  1,
            trigger            :  1, /* 0: edge, 1: level */
            mask         :  1, /* 0: enabled, 1: disabled */
            __reserved_2 : 15;

      union {           struct { __u32
                                 __reserved_1 : 24,
                                 physical_dest      :  4,
                                 __reserved_2 :  4;
                        } physical;

                        struct { __u32
                                 __reserved_1 : 24,
                                 logical_dest :  8;
                        } logical;
      } dest;

} __attribute__ ((packed));
```

Par exemple, pour programmer le IO_APIC de sorte qu'il délivre l'interruption 4 au premier processeur dans un système multiprocesseurs, il suffit de modifier la fonction *setup_IO_APIC_irqs()*, qui initialise le IO_APIC dans le programme /usr/src/linux/arch/i386/kernel/io_apic.c

```
void __init setup_IO_APIC_irqs(void)
{     .....
            /* Is this the configuration of the IRQ 4*/
            if (pin==4) {
                  /* Fixed delivery Mode of interrupt as regular IRQ */
                  entry.delivery_mode = 0;
                  entry.dest_mode = 0;
                  entry.mask = 0;                        /* enable IRQ */
                  /* We will broadcast the interrupt to all CPUs*/
                  entry.dest.physical.physical_dest = 15; /* 4 bits du
*registre ID du local APIC sont utilisés pour désigner le(s) processeur(s)
*qui vont desservir l'interruption. Dans notre cas la destination est tous
*les microprocesseurs donc la valeur 15 sera attribué à ce registre
*/
            }
      .....
```

La documentation d'Intel [16], contient plus de détails sur la programmation du IO_APIC [17].

Architecture Itanium :

Avec cette nouvelle architecture de processeurs, Intel a mis en place un système de gestion d'interruption SAPIC [18] (Streamlined Advanced Programmable Interrupt Controler). Lorsque Intel a mis en place le système IO_APIC, il fallait qu'il soit compatible avec l'ancien système PIC 8259A. Avec la mise en place d'un nouveau système, Intel a saisi l'occasion de mettre en place un nouveau système formé en deux parties non compatible avec le traditionnel PIC 8259A. La première partie est I/O SAPIC, connectée aux périphériques, et la deuxième partie est LSAPIC, qui se charge de la livraison des interruptions au processeur local. Les deux parties communiquent à travers un système de bus. Le SAPIC a un temps de réponse aux interruptions rapide et peut supporter jusqu'à 256 microprocesseurs, au lieu de 16 comme les deux autres PIC.

2.1.3 Gestionnaire d'Interruption

Une fois que le microprocesseur confirme qu'il va traiter la demande d'interruption, il lui faut chercher l'adresse du gestionnaire d'interruption qu'il doit exécuter pour servir la demande. Cette adresse est stockée dans une table de description d'interruption IDT, qui est pointée par le registre idtr du microprocesseur. Cette table est formée de 256 entrées, une entrée pour chaque vecteur d'interruption ou exception. Chaque entrée est

un descripteur de 8 octets contenant l'adresse du gestionnaire d'interruption ou d'exception correspondant.

Pour servir une demande d'interruption, le microprocesseur doit donc d'abord déterminer le vecteur i associé avec la demande d'interruption, puis lire l'entrée i dans la table IDT, pour chercher l'adresse du gestionnaire d'interruption correspondant, puis faire d'autres traitements tel que s'assurer que la demande d'interruption est issue d'une source autorisée, et finalement basculer vers le code du gestionnaire d'interruption.

La gestion des interruptions est l'une des tâches les plus sensibles effectuées par le noyau Linux, car elle doit satisfaire aux contraintes suivantes :

- les interruptions peuvent arriver à n'importe quel moment, tandis que le noyau est en train de finir l'exécution d'une autre routine. Le but du noyau est de servir et d'en finir avec le traitement de l'interruption le plus tôt possible, pour revenir terminer ce qu'il était en train d'exécuter;
- les gestionnaires d'interruptions doivent être codés de manière à ce que leur exécution puisse être imbriquée. En effet, les interruptions peuvent arriver à n'importe quel moment, et il se peut que le noyau soit en train d'exécuter le code du gestionnaire d'interruption d'une demande d'interruption au moment où une nouvelle demande d'interruption arrive;
- les régions critiques du noyau où les interruptions sont désactivées doivent être très courtes pour satisfaire le point précédent.

Le gestionnaire d'interruption effectue une série de tâches qui n'ont pas la même priorité d'exécution. Les tâches longues et non critiques par rapport au gestionnaire d'interruption doivent être exécutées plus tard car, au moment de l'exécution du gestionnaire d'interruption, les demandes d'interruption de la ligne en question sont

ignorées temporairement, et il se peut qu'au moment de l'arrivée de la demande d'interruption le microprocesseur servait une autre demande d'interruption. Plus important, si le gestionnaire d'interruption a interrompu un processus, ce dernier doit rester toujours dans l'état TASK_RUNNING, sinon le système gèlera. Par conséquent, le gestionnaire d'interruption ne peut effectuer des opérations bloquantes telles que l'accès au disque dur. Pour le gestionnaire d'interruption, Linux distingue trois types de tâches. Les *tâches critiques* (e.g. la re-programmation du PIC) et *non critiques* (e.g. lecture des coordonnées de la souris) se placent dans le code du gestionnaire d'interruption, tandis que les tâches *non critiques non urgentes* (e.g. copier un ensemble de données vers l'espace d'un autre processus) sont exécutées plus tard, à l'aide de fonctions spécifiques appelées "bottom half".

La figure 2.3 schématise la gestion des interruptions sous Linux.

Le code de la routine IRQn est le suivant :

```
IRQn_interrupt :
      pushl $n-256
      jmp common_interrupt
```

et common_interrupt est représenté par le code suivant :

```
      common_interrupt :
            SAVE_ALL
            call do_IRQ
            jmp $ret_from_intr
```

La macro SAVE_ALL permet de sauvegarder les registres du microprocesseur. Après cela, c'est la fonction do_IRQ qui va faire tout le travail. Cette fonction est située dans le fichier /usr/src/linux/arch/i386/kernel/irq.c.

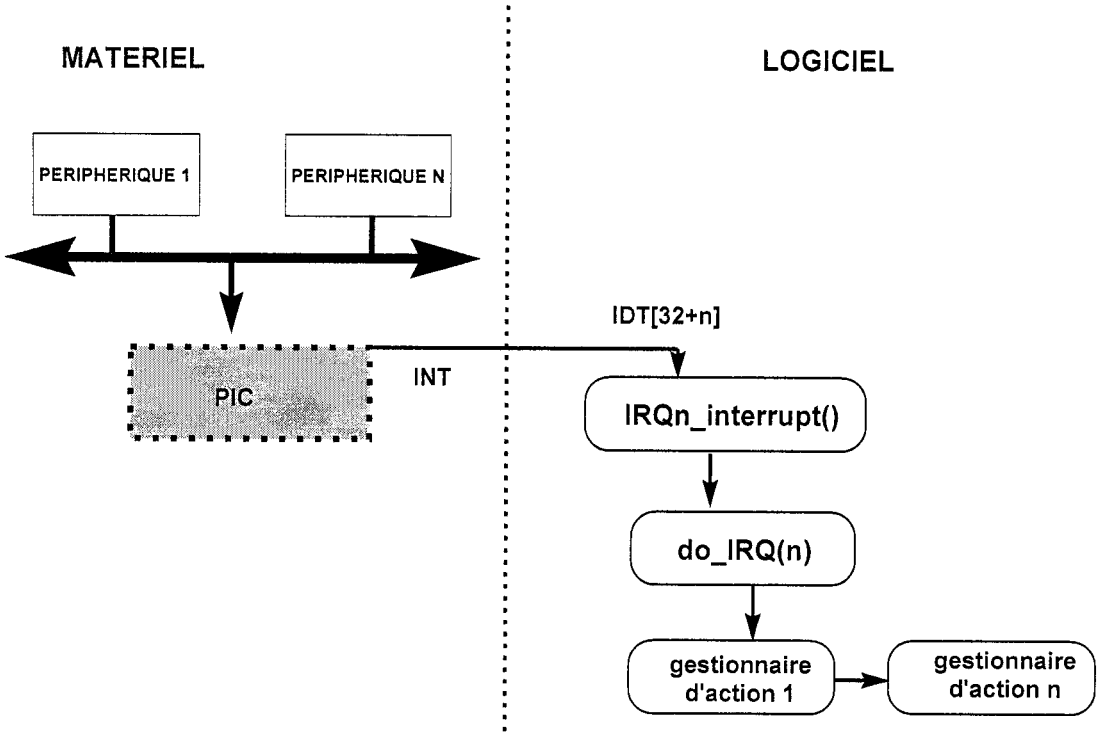

FIGURE 2.3 : Gestion des interruptions sous Linux.

D'une manière abstraite, la fonction détermine premièrement la ligne de demande d'interruption IRQ en question, puis effectue d'autres traitements (acquérir le verrou associé à cette ligne IRQ, envoyer une notification de la réception de la demande d'interruption au PIC ...). Elle appelle enfin le gestionnaire d'action approprié, associé à cette ligne IRQ, dont la tâche est d'effectuer les traitements nécessaires pour répondre à la demande d'interruption (ex : lecture des nouvelles coordonnées de la souris).

2.2 Référence du temps

La nature des expériences faites dans le cadre de ce mémoire exige l'utilisation d'une source de temps externe précise et fiable. Elle va constituer notre référence du temps, afin d'effectuer des lectures du registre compteur de cycles d'horloge et de mesurer le temps de transfert dans un réseau Ethernet. Plusieurs solutions existent. L'utilisation d'un récepteur GPS qui, une fois branché à son antenne, peut délivrer un signal à chaque seconde (PPS) avec une précision de ± 12 nanosecondes, a été retenue ici.

2.2.1 GPS

L'acronyme GPS veut dire système de positionnement global. Ce système qui est aussi référé par NAVSTAR (NAVigation Satellite Timing and Ranging) a nécessité un investissement de $12 milliards de dollar par le département de la défense Américaine.
Le concept du système est de mettre en orbite 24 satellites. Ces derniers sont contrôlés par des stations situées à différents endroits sur la terre, et reçoivent périodiquement de ces dernières les données *ephemeris* et *alamanc* (comme expliqué par la suite). Les satellites sont mis sur six différents chemins orbitaux, sont constamment en mouvement par rapport à la surface et effectuent deux fois le tour de la terre en vingt quatre heures. Ce système permet aux utilisateurs, peu importe leur position sur le globe ou l'état de la météo, de recevoir les signaux envoyés par les satellites sur leur récepteur GPS. Ils peuvent ainsi déterminer le temps, leur vitesse de déplacement et leur position en trois

dimensions selon le nombre de satellites visibles par le récepteur GPS. Avec 3 satellites le récepteur GPS peut déterminer la position en 2 dimensions latitude et longitude. Avec quatre satellites ou plus, le récepteur GPS peut déterminer la position en trois dimensions latitude, longitude et altitude.

Les signaux envoyés par les satellites contiennent un pseudo-code aléatoire formé de données *ephemeris* et *almanac*.

Les données *ephemeris* [19] contiennent des informations comme l'état de fonctionnement du satellite, la date courante et l'heure. Les satellites du système GPS sont équipés d'horloges atomiques pour maintenir le temps, ce qui permet aux récepteurs GPS d'avoir le temps synchronisé d'une manière atomique. Le temps GPS [20] était 0 à la date 0h 6-Jan-1980, et depuis ce temps il n'a jamais été perturbé par la seconde introduite dans le système de temps UTC. L'horloge GPS est actuellement en avance de 13 secondes par rapport au temps UTC.

Les données *almanac* permettent aux récepteurs GPS de savoir la position de n'importe quel satellite dans l'orbite à n'importe quel moment.

La figure 2.4 schématise le concept du système GPS

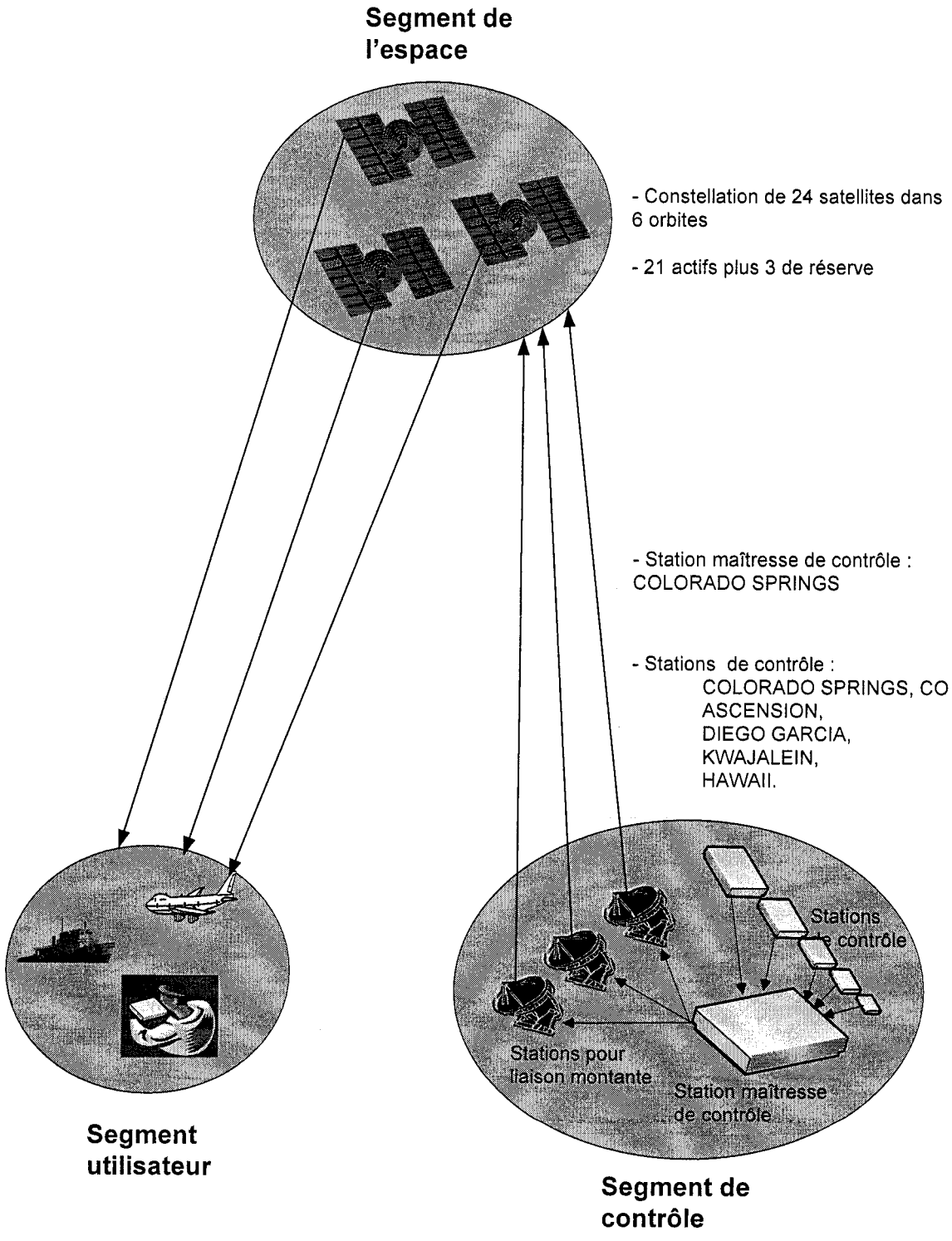

FIGURE 2.4 : Système de positionnement global GPS

2.3 Gestion des réseaux sous LINUX

Linux supporte différents types d'architecture réseau. Le code actuel de gestion des réseaux est inspiré du code original du système Unix BSD. Le code source est organisé en couches avec une interface de communication bien définie entre chacune de ces dernières.

Pour rendre la gestion des réseaux plus efficace, Linux évite de faire transiter les paquets d'informations d'une couche à l'autre, en mettant les données originales dans un tampon mémoire suffisamment grand afin de permettre aux autres couches d'ajouter leur information de contrôle.

L'interface de programmation d'application API de Linux est basée sur les sockets de Unix BSD. En fait, un socket est une communication entre deux points terminaux. Il permet la transmission d'information d'un point vers un autre. Ils sont implémentés sous Linux comme des fichiers appartenant au système de fichier sockfs.

Notre étude concerne les réseaux TCP/IP, qui sont composés en quatre couches. Le tableau 2-1 schématise les couches de réseaux avec des exemples d'application à chaque niveau.

TABLEAU 2.1 : Architecture du réseau TCP/IP

Couche Application	FTP	SMTP	HTTP	...
Couche Transport	TCP	UDP	SCTP	...
Couche Network	IP	IPv6	IPX	...
Couche liaison de données	Ethernet	Token ring	FDDI	...

L'envoi d'un paquet d'informations d'un nœud à un autre dans le réseau se passe en deux étapes.

La première étape est au niveau du nœud lui même. Le paquet transite de la couche application jusqu'à la couche liaison de données, en passant par les autres couches intermédiaires. Les fonctionnalités de ces couches sont implantées au niveau du système d'exploitation Linux. Cette transition nécessite un temps qui est sujet à des variations selon la charge du système. La deuxième étape consiste en la transmission réelle du paquet dans le réseau pour qu'il atteigne le nœud de destination.

L'objectif ici est de caractériser jusqu'à quel degré le temps d'aller et de retour d'un message entre deux nœuds est symétrique, puis jusqu'à quel point un paquet diffusé sur le réseau atteint tous les nœuds au même moment. Pour mesurer ce temps avec précision, il faudra mesurer le temps pris dans l'étape 2, et non celui de l'étape 1, afin d'éviter l'inexactitude des mesures entraînée par la latence variable causée par le système d'exploitation.

Pour ce faire, lorsque le nœud émetteur envoie le paquet, le temps d'envoi sera inséré au dernier moment avant que le paquet ne soit transmis réellement par la carte réseau. De même, lorsque le paquet sera reçu par le nœud destinataire, le temps d'arrivée sera lu aussitôt que le paquet sera disponible dans la première couche du modèle TCP/IP, qui est la couche liaison de données. Ainsi, il faudra modifier le code source du pilote de la carte réseau, plus précisément les fonctions qui se chargent de l'envoi et de la réception de paquets directement de la carte réseau.

Notons que les équipements utilisés pour l'interconnexion des nœuds ont un impact sur le délai de transmission réseau. Ils peuvent être soit un concentrateur, soit un commutateur ou encore un routeur.

Concentrateur : Il renvoie les données qu'il reçoit d'un émetteur sur toutes ses sorties. Ainsi, le destinataire approprié va traiter le message, et les autres nœuds vont ignorer le message.

Commutateur : Au début, il fait le même travail que le concentrateur. Au fur et à mesure, il apprend la localisation des nœuds pour ensuite n'acheminer le message provenant d'un nœud émetteur que vers la sortie correspondante au nœud destinataire. Ceci réduit le nombre de paquets retransmis à chaque sortie.

Routeur : C'est une sorte d'ordinateur spécifique pour les réseaux. En plus d'acheminer le paquet vers sa destination, il peut traiter les paquets qu'il reçoit au niveau de la couche réseau.

Chapitre 3

Architecture du système de mesure

Ce chapitre discute du principe global des programmes tests qui vont permettre d'étudier le registre compteur de cycles d'horloge et le délai de transmission dans les réseaux Ethernet.

3.1 Registre compteur de cycles d'horloge

L'objectif est de mesurer la précision de l'horloge du microprocesseur, ainsi que sa déviation au cours du temps. Pour ce faire, un récepteur GPS (Motorola M12+ Timing Oncore) est connecté au port série de l'ordinateur. Ce dernier constitue notre référence du temps grâce à sa capacité de délivrer un signal par seconde (PPS) avec une précision de ± 12 nanosecondes. Ce signal est délivré à travers le signal DCD du connecteur série DB9. Un câble male / femelle RS-232 spécifique a été conçu, ne reliant que le

connecteur DCD et la masse, afin de ne recevoir du récepteur GPS que le signal PPS, et du même coup réduire le bruit qui peut être causé par les autres signaux qui parviennent de ce dernier.

La valeur du registre compteur de cycles d'horloge est lue à la réception de chaque signal PPS du récepteur GPS. Par la suite, la différence entre deux lectures successives est calculée pour déterminer combien de cycles d'horloge se sont écoulés durant cette seconde. Autrement dit, cette différence observée représente la fréquence d'horloge du microprocesseur. Ainsi, à la fin de la période du test, ces différences vont former le fichier de données qui sera analysé pour calculer différentes valeurs statistiques (moyenne, écart type...), ce qui va permettre de discuter de la précision et de la stabilité de l'horloge du microprocesseur.

La nature des tests qui doivent être effectués oblige à diviser le programme de test en trois composantes, afin d'obtenir des lectures exactes. La figure 3.1 schématise l'environnement expérimental et la figure 3.2 illustre les trois composantes ainsi que le fonctionnement global du programme.

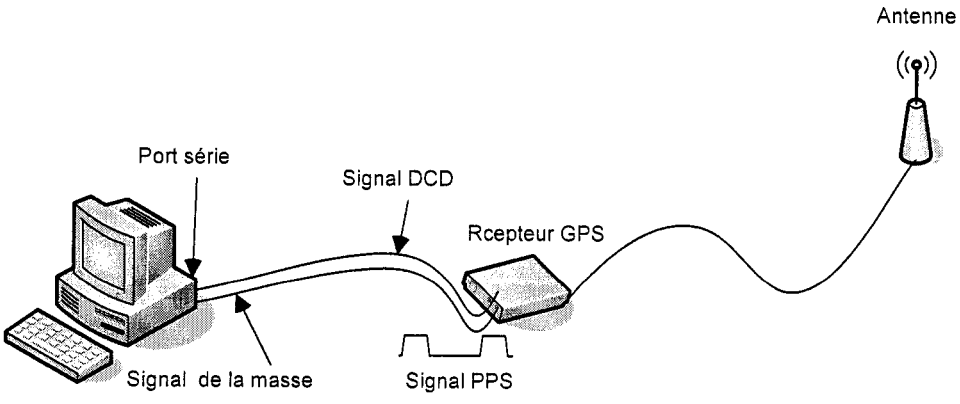

FIGURE 3.1 : Environnement expérimental

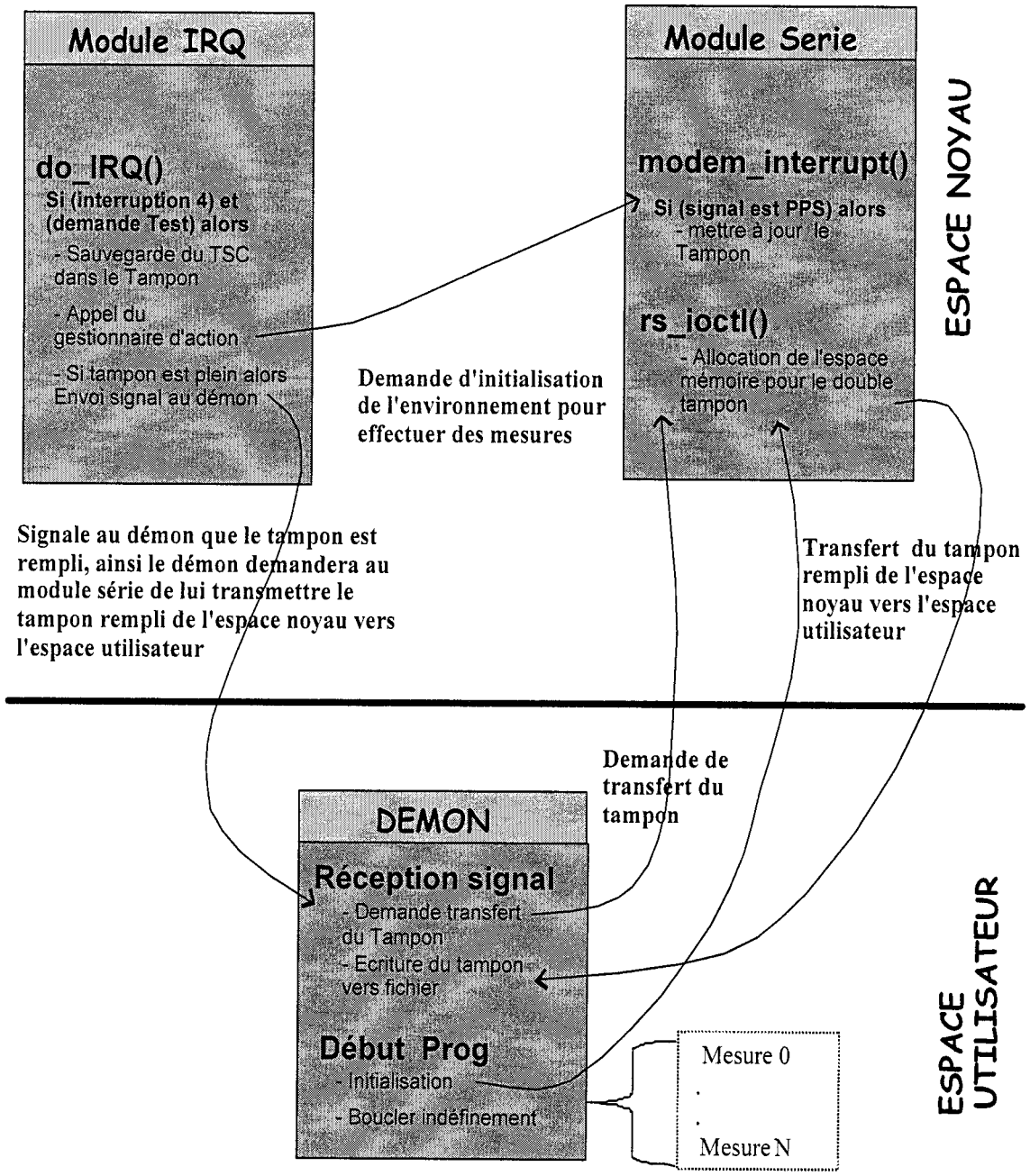

FIGURE 3.2 : Principe global de fonctionnement du programme de test

3.1.1 Module IRQ

Ce module se charge de la gestion des interruptions et il est dépendant de l'architecture sur laquelle Linux tourne. La fonction *do_IRQ()* expliquée dans le chapitre précédent est implémentée dans le fichier /usr/src/linux/arch/i386/kernel/irq.c Elle est invoquée dès que Linux accepte de traiter une demande d'interruption. Ainsi, pour obtenir des lectures précises du registre compteur de cycles d'horloge, le code source de cette fonction est modifié pour effecteur des lectures du registre dans celle-ci.

Dans l'architecture Intel, par exemple, le port série se voit attribuer la ligne de demande d'interruption 4. La fonction *do_IRQ()*, détermine la ligne IRQ qui a fait la demande d'interruption par le code suivant : `int irq = regs.orig_eax & 0xff;`. Juste après cette ligne, le test sera effectué pour déterminer si la demande d'interruption provient du port série. Si c'est le cas, la lecture du registre compteur de cycles d'horloge sera faite, puis sa valeur sera sauvegardée dans un tampon mémoire double.

Notons que l'interruption sur le port série s'active par la montée ou la descente de la pente du signal. Tel que montré à la figure 3.3, pour chaque signal PPS deux interruptions vont être acheminées vers le port série. La détermination exacte de la pulsation qui nous intéresse, qui correspond à l'arrivée du signal PPS, sera faite dans le module série, qui contient le gestionnaire d'action approprié du port série. D'après les expériences effectuées la durée de la phase d'activation de l'interruption est inférieure à 500 nanosecondes et on peut assumer que cette durée est constante.

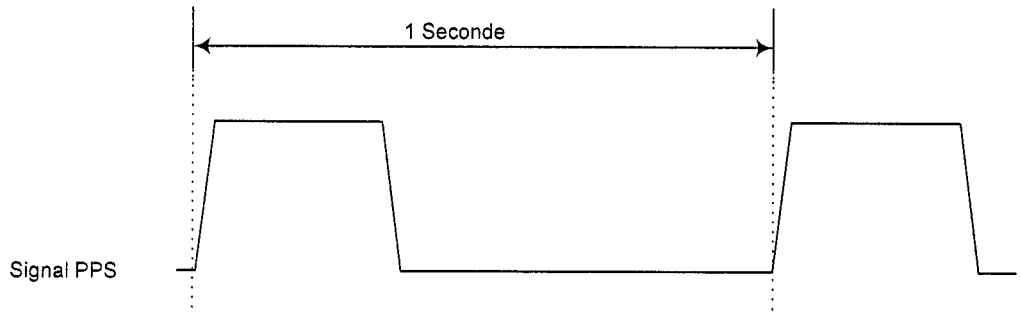

FIGURE 3.3 : Signal PPS du récepteur GPS

Après l'exécution du gestionnaire d'action et avant d'en finir avec la fonction *do_IRQ()*, le tampon est vérifié, s'il est plein la fonction le signale au démon, pour qu'il demande au module série de lui transmettre les données de l'espace noyau vers l'espace utilisateur.

La structure de données utilisée pour contenir les lectures du registre compteur de cycles d'horloge est la suivante :

```
typedef struct {
        unsigned long *gltt_tsc_low, *gltt_tsc_low_head, *gltt_tsc_high,
                      *gltt_tsc_high_head;
        cycles_t *gltt_tsc, *gltt_tsc_head;
        unsigned long *gltt_sec, *gltt_usec;
        unsigned long *gltt_sec_head, *gltt_usec_head;
        unsigned long gltt_pps_read;
        short int gltt_cpu_pass;
        short int *gltt_is_this_pps, *gltt_is_this_pps_head;
        struct timeval gltt_pc_time;
} t_ltt_sample;
```

3.1.2 Module série

Ce module a deux rôles. Le premier est d'agir comme intermédiaire entre le module IRQ et le démon. Le deuxième est de déterminer si l'interruption en cours correspond au signal PPS.

Pour le premier rôle, à la demande du démon, il va réserver l'espace mémoire du double tampon dans l'espace noyau pour la sauvegarde des lectures du registre compteur de cycles d'horloge, ainsi qu'initialiser d'autres variables d'environnement pour le déroulement du test (utilisation de l'instruction *rdtsc()* ou la macro *get_cycles()* pour la lecture du registre).

Pour le deuxième rôle, sachant que pour chaque signal PPS deux interruptions parviennent sur le port série, il faut déterminer laquelle des deux correspond au signal PPS. Le code du gestionnaire d'action du module série est modifié de sorte que, dès qu'il détermine que l'interruption correspond au signal PPS, il va mettre à jour le tampon mémoire utilisé pour le test, pour indiquer que cela est un signal PPS. Cela se fait en testant les registres du contrôleur du port série, l'UART.

Notons que le module série se situe dans l'espace noyau et que le tampon utilisé pour les tests est déclaré global, le module a donc un accès direct à ce tampon. Par contre, la communication entre le démon, qui se trouve dans l'espace utilisateur, et le module série se fait à l'aide de la fonction de contrôle d'entrée/sortie *ioctl()*.

3.1.3 Démon

Il constitue le point de départ des tests car, une fois lancé, il initialise l'espace mémoire nécessaire pour contenir les données qui lui seront transmises de l'espace noyau. Puis, il demande au module série de réserver le double tampon dans l'espace noyau, ainsi que d'initialiser des variables d'environnement nécessaires pour le déroulement du test. Par la suite, il configure sa fonction qui intercepte les signaux provenant du noyau. Enfin, il entre dans une boucle infinie.

Une fois le signal envoyé du module IRQ au démon, la fonction appropriée s'active. Elle demande au module série de lui transférer le contenu du tampon plein puis elle écrit ces données dans un fichier de sortie.

3.2 Réseau Ethernet

Deux tests ont été effectués afin de calculer le temps de transmission sur le réseau et de déterminer jusqu'à quel point un message diffusé parvient aux nœuds du réseau au même moment. Ces tests mettent en jeu plusieurs nœuds dont les horloges ne sont pas synchronisées. Même si elles étaient synchronisées au départ, il y aura toujours une déviation d'horloge d'un nœud par rapport à l'autre au cours du temps, d'où la nécessité d'avoir une référence de temps. Pour cela, un câble en Y est branché au connecteur série du récepteur GPS, puis les deux autres extrémités du câble sont branchées aux ports séries des ordinateurs qui sont utilisés pour l'expérience.

3.2.1 Temps de transmission

Le programme pour mesurer le temps de transmission réseau se divise en deux parties. La première partie comprend les programmes client et serveur qui tournent sur deux nœuds différents. Le client envoie un message au serveur, puis ce dernier réplique par un autre message. Finalement, le client enregistre cet autre message dans un fichier de sortie. Ces derniers s'échangent des messages pour calculer le temps aller et retour dans le réseau. La taille du paquet échangé entre les deux nœuds est la même, même si au début de la communication le paquet ne contient que le temps estampillé du côté client. En effet, la taille du paquet envoyé influence le délai de transmission.

La deuxième partie consiste en la modification du code source du module qui se charge de la gestion de la carte réseau, pour estampiller les paquets que le client et le serveur s'échangent. Le but est d'estampiller le paquet entrant dès son arrivée, et d'estampiller le paquet sortant au dernier moment, c'est à dire juste au moment où il va être transféré vers la carte réseau. De cette manière, on mesure seulement le temps de transfert dans le réseau, en éliminant celui du traitement du paquet par Linux.

Pour localiser les points où le temps doit être estampillé, il faut déterminer les premières lignes du code qui sont exécutées lorsqu'un paquet arrive sur la carte réseau, et celles qui sont exécutées lorsqu'un paquet va être acheminé vers la carte réseau.

Il faut d'abord déterminer la fonction qui joue le rôle du gestionnaire d'action, le code du programme qui s'exécute lorsqu'une interruption de la carte réseau se produit. Dans les tests effectués, une carte « 3Com Corporation 3c905B 100BaseTX » a été utilisée. Le

code source du module gérant celle-ci est /usr/src/linux/drivers/net/3c59x.c. Dans la fonction d'ouverture du module, le gestionnaire d'action est représenté par la fonction *boomrang_interrupt()*. Lorsqu'un paquet arrive, les lignes suivantes du gestionnaire d'action s'exécutent.

```
static void boomerang_interrupt(int irq, void *dev_id, struct pt_regs
*regs)
{
.....
                if (status & UpComplete) {
                        rdtsc(ltt_in_tsc_low, ltt_in_tsc_high);
                        outw(AckIntr | UpComplete, ioaddr + EL3_CMD);
                        if (vortex_debug > 5)
                                printk(KERN_DEBUG "boomerang_interrupt-
>boomerang_rx\n");

                        boomerang_rx(dev);
                }
.....
}
```

C'est ici que le temps auquel le paquet est arrivé sera estampillé.

Pour l'envoi d'un paquet, Linux utilise la fonction 'start_xmit', qui se charge du transfert réel du paquet sur la carte réseau. En fait, c'est un pointeur de fonction. Dans notre cas, la fonction d'initialisation du module de notre carte réseau associe le pointeur de la fonction start_xmit à la fonction *boomerang_start_xmit ()*. Il faut simplement modifier le code de cette dernière pour estampiller le paquet avant son envoi.

3.2.2 Temps de diffusion

Il est intéressant de mesurer le temps de diffusion, plus précisément de déterminer jusqu'à quel point le temps que prend un message diffusé pour atteindre tous les nœuds

du réseau varie. Cette mesure se divise en deux parties. La première partie comprend le programme émetteur qui diffuse continuellement des messages numérotés UDP sur le réseau et le programme récepteur de chaque noeud, qui interceptent ces messages pour écrire le paquet estampillé reçu dans un fichier de sortie.

La deuxième partie est semblable au test précédent, sauf qu'ici on s'intéresse au temps estampillé à la réception du paquet diffusé dans chaque nœud.

Chapitre 4

Résultats

Après avoir vu dans le chapitre précédent les algorithmes et les structures de données qui ont été utilisés pour faire nos tests, ce chapitre présente les résultats obtenus afin de caractériser la précision et la stabilité de la fréquence de l'horloge des microprocesseurs, ainsi que la variabilité du temps de transmission dans les réseaux Ethernet.

4.1 Registre compteur de cycles d'horloge

Dans cette suite de tests, l'objectif est de mesurer la précision de l'horloge du microprocesseur, ainsi que sa déviation au cours du temps. Avant de conduire les tests complets, il faut en déterminer la durée. En fait, le nombre d'observations requises pour l'étude d'un phénomène dépend de sa durée de vie. Dans notre cas, la durée de vie d'un microprocesseur dépasse facilement les trois années, ce qui implique une longue durée de test, qui ne peut convenir dans notre cas.

Allan [25] [26] a proposé une méthode qu'il a nommé racine d'Allan pour étudier la stabilité de la fréquence des oscillateurs, ce qui va nous aider à formuler un critère pour la durée des tests. La méthode est basée sur la suite des moyennes mobiles des déviations de la fréquence dans le temps.

Si l'oscillateur est stable après un certain nombre de lecture de sa fréquence, les lectures qui suivent n'auront presque plus d'impact sur les valeurs statistiques concernant les observations effectuées.

Deux séries de mesures sont effectuées (une heure et quatre heures) plusieurs fois, afin de mieux comprendre le phénomène et pouvoir statuer sur la durée des tests pour le reste des expériences.

Les graphes qui suivent sont produits par le logiciel AlaVar [25].

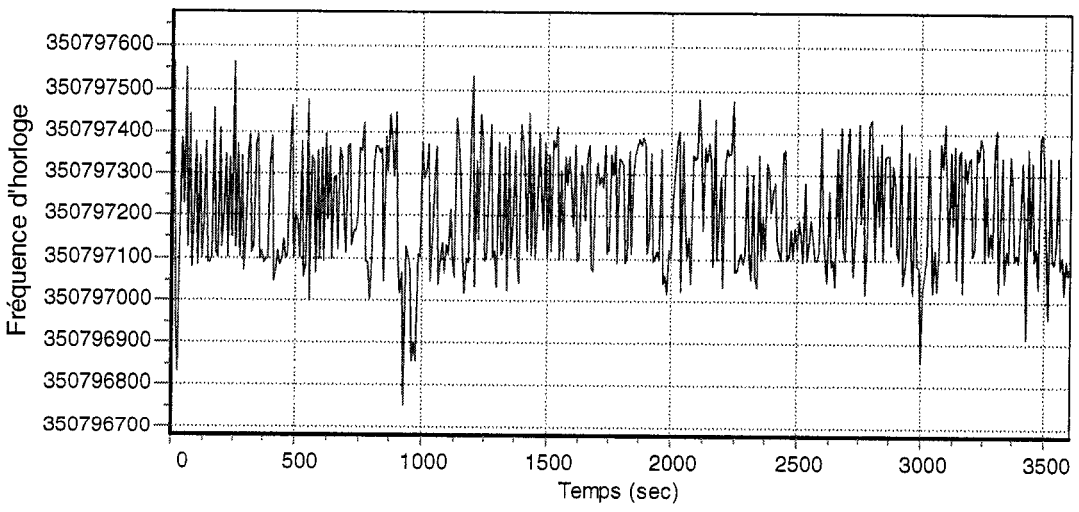

FIGURE 4.1 : Fréquence d'horloge du microprocesseur sur une durée d'une heure

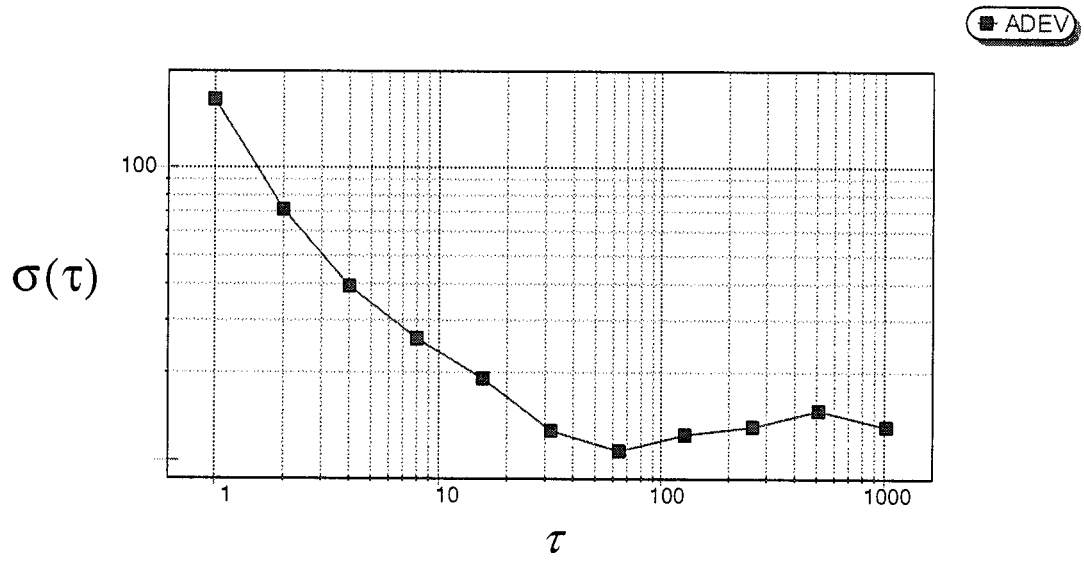

FIGURE 4.2 : Variance de la racine d'Allan de la fréquence d'horloge du microprocesseur sur une durée d'une heure.

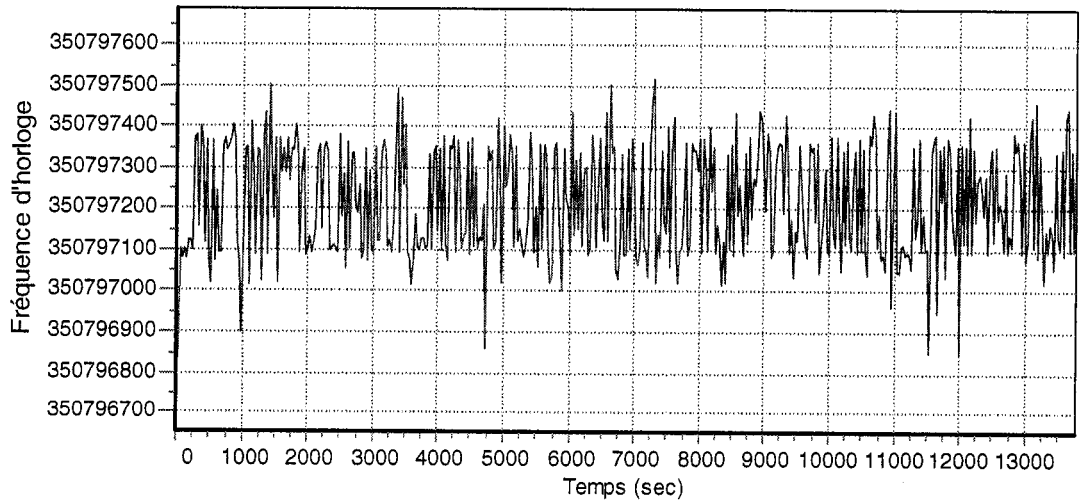

FIGURE 4.3 : Fréquence d'horloge du microprocesseur sur une durée de 4 heures.

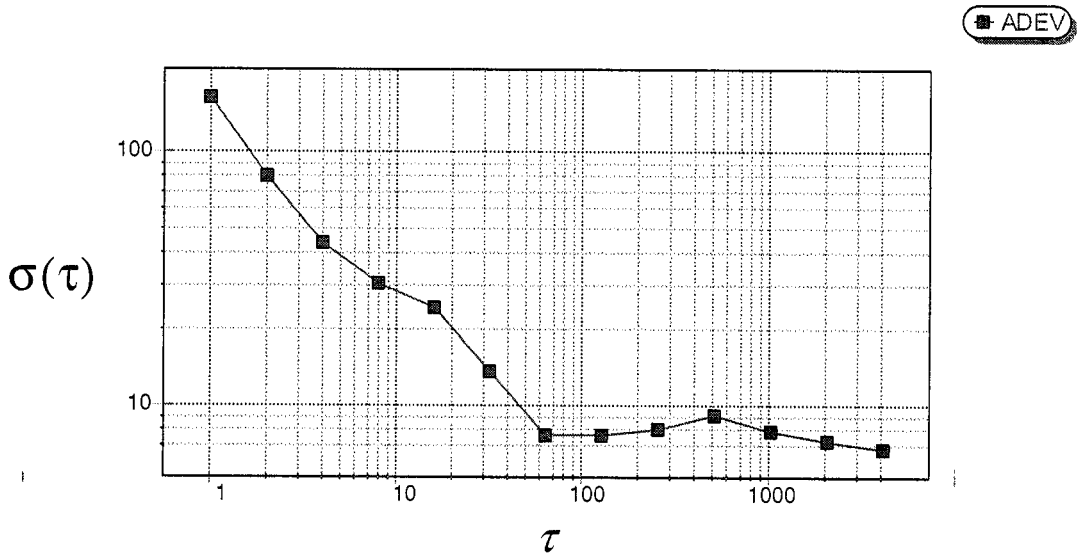

FIGURE 4.4 : Variance de la racine d'Allan de la fréquence d'horloge du microprocesseur sur une durée de 4 heures.

Pour les deux durées de test, on constate que la variance de la racine d'Allan converge vers un état stable à partir de la valeur 64 de l'ordre de la moyenne mobile. Ainsi, moins de 100 observations peuvent suffire pour faire nos tests.

Le tableau 4.1 résume le sommaire des statistiques des deux expériences.

TABLEAU 4.1 : Sommaire des statistiques de la fréquence d'horloge du microprocesseur sur une durée d'une heure et de quatre heures.

	100 observations	1 Heure	4 Heures
Moyenne (cycle horloge)	350 797 135	350 797 217	350 797 208
Écart type	157,76	139,02	134,55
Médiane	350 797 115	350 797 177	350 797 154
Minimum	350 796 716	350 796 679	350 796 652
Maximum	350 797 787	350797682	350797688
Étendue des valeurs	1 071	1 003	1 036

D'après le tableau ci-dessus, on constate que les valeurs statistiques pour les deux durées de test, 1 heure et 4 heures, sont presqu'identiques à celles obtenues lorsqu'on ne prend en compte que 100 observations.

En définitive, les deux approches nous montrent qu'une durée de test d'une heure est amplement suffisante pour étudier la précision et la stabilité de l'horloge du microprocesseur. Bien que 100 observations puissent suffire, on prend une durée d'une heure pour mieux valider les résultats obtenus.

L'écart type permet de quantifier la dispersion des données autour de la moyenne. Dans le cas présent, cela permet de déterminer l'erreur de précision de l'horloge du microprocesseur avec un certain niveau de confiance, selon la loi (normale, logarithmique...) avec laquelle les observations peuvent être approchées. D'après le tableau 4.1, on peut dire que, pour la durée d'une heure de test, l'horloge du microprocesseur testé a une fréquence de 350797217 Hz ± 139 cycles horloge. Cependant, la nature de ces observations laisse cette estimation avec niveau de confiance décevant de 76.9 %.

L'analyse des observations obtenues avec le logiciel de statistique [32] a démontré qu'elles ne peuvent être approchées par la loi normale. Ainsi, on se propose d'effecteur des calculs pour déterminer l'erreur de précision de l'horloge du microprocesseur qui correspond à un niveau de confiance égale à 99%. Pour y parvenir, on varie un coefficient multiplicateur de l'écart type, puis on détermine combien d'observations sont incluses dans le nouvel intervalle de confiance, déterminé par la valeur moyenne de l'horloge du microprocesseur ± (coefficient * écart type). Par la suite, on divise le

nombre d'observations obtenues par le nombre total des observations de l'expérience, afin d'obtenir le niveau de confiance correspondant. On répète le même processus jusqu'à ce qu'on atteigne un niveau de confiance égale à 99% et c'est ainsi qu'on obtiendra l'erreur de précision correspondante déterminée par (coefficient * écart type).

L'horloge du microprocesseur est basée sur un cristal en quartz dont la fréquence varie suite à une variation de sa température. Le premier genre de test à faire est donc de varier la température afin d'observer son impact sur la fréquence d'horloge du microprocesseur. Une autre variante de ce type de test consiste à faire varier la charge de travail du microprocesseur pour constater l'impact que cela peut avoir. En effet, l'augmentation de la charge de travail du microprocesseur a pour effet d'augmenter la température graduellement et d'une manière naturelle de ce dernier.

Avant d'en finir avec ces tests, différentes architectures et générations de microprocesseurs seront étudiées pour déterminer l'impact de ce paramètre sur l'horloge des microprocesseurs.

La version 2.4.26 du noyau Linux est utilisée dans tous les tests avec les machines en état de repos (aucun programme ne tourne).

4.1.1 Variation de la température

Dans ce test, la température du microprocesseur est augmentée jusqu'à un niveau qui ne peut être atteint normalement même lorsque le microprocesseur est en plein régime de fonctionnement.

Le cobaye de cette expérience est un ordinateur doté d'un microprocesseur Pentium II. Sa ventilation a été arrêtée, puis l'expérience a commencé.

Pour la lecture de la température, le logiciel *lm-sensor* [33] a été utilisé. Une fois compilé et installé, ce programme détecte les capteurs de températures présents dans l'ordinateur. Le capteur W83781D contrôlant la température du microprocesseur est celui qui importe ici. La température courante est obtenue avec la commande suivante : "#cat /proc/sys/dev/sensors/w83781d-i2c-0-2d/temp". Cette expérience exige beaucoup d'attention, afin de ne pas griller le microprocesseur ou autre circuit de l'ordinateur.

La température de début de l'expérience est 26 °C et l'expérience est arrêtée à 48,5 °C.

TABLEAU 4.2 : Sommaire des statistiques de la fréquence du microprocesseur en fonction de la température

	INTERVALLE DE TEMPERATURE (°C)					
	26-30	30-34	34-38	38-42	42-46	46 - 48,5
ECART TYPE σ (cycle horloge)	464,67	1 686,58	1 878,87	2 299,72	1 469,95	1 929,43
MOYENNE $\overline{X}$ (cycle horloge)	349 205 333	349 204 698	349 204 068	349 203 314	349 202 799	349 202 435
INTERVALLE DE CONFIANCE A ≈ 99 %	$\overline{X} \pm 2{,}08\ \sigma$	$\overline{X} \pm 5{,}94\ \sigma$	$\overline{X} \pm 6{,}89\ \sigma$	$\overline{X} \pm 6{,}05\ \sigma$	$\overline{X} \pm 6{,}68\ \sigma$	$\overline{X} \pm 5{,}78\ \sigma$
ERREUR DE PRECISION (cycle horloge) A ≈ 99 %	967	10 018	12 964	13 936	9 819	11 191
ERREUR DE PRECISION (µs) A ≈ 99 %	2,768	28,689	37,125	39,909	28,119	32,047

Dans la figure 4.5, il apparaît clairement que plus la température augmente, plus la moyenne de fréquence d'horloge du processeur diminue. Cette diminution correspond à une valeur approximative de 600 cycles d'horloge entre chaque intervalle de température de 4 °C.

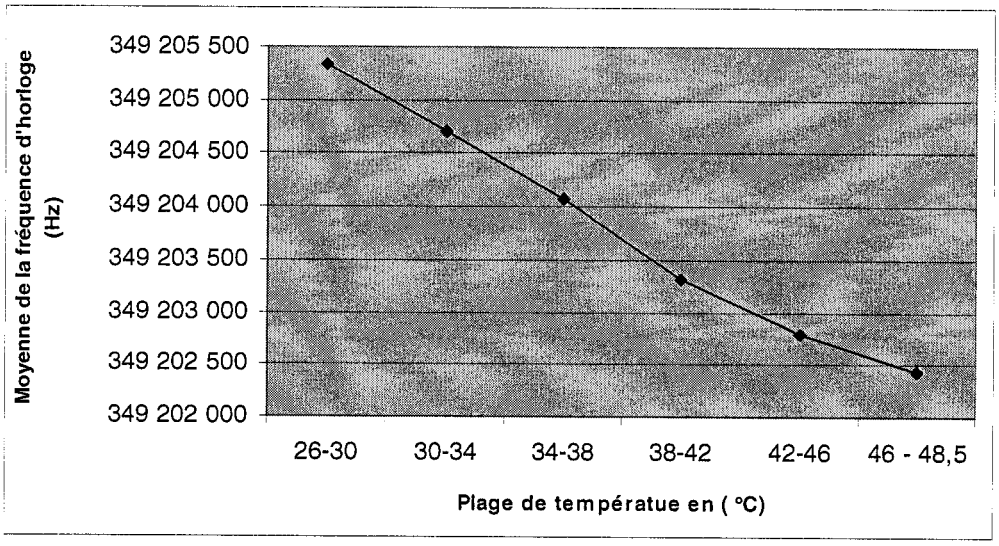

FIGURE 4.5 : La moyenne de la fréquence d'horloge du microprocesseur par intervalle de température.

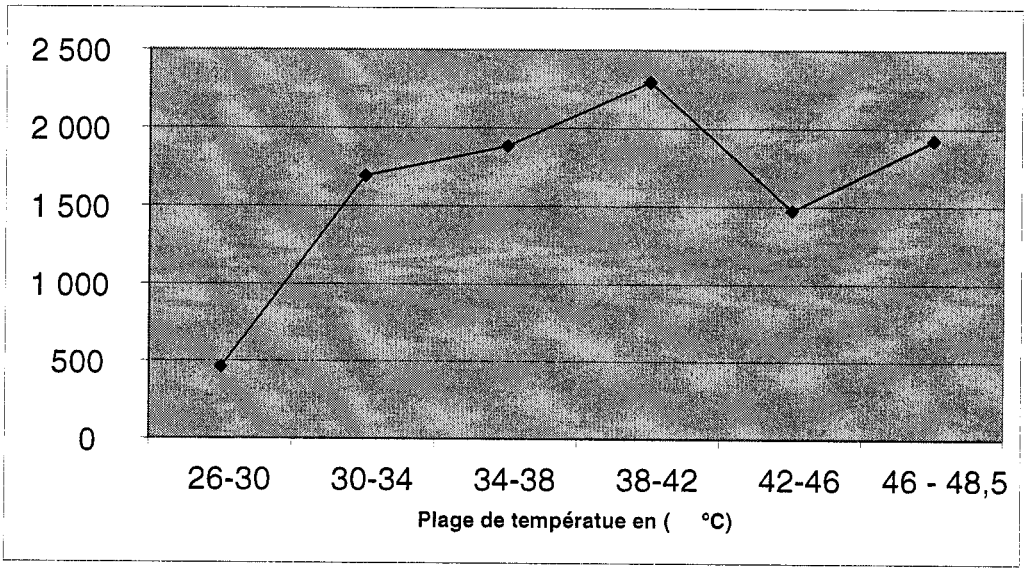

FIGURE 4.6 : L'écart type de la fréquence d'horloge par intervalle de température.

Pour un niveau de confiance de 99 %, on a une erreur de précision qui augmente considérablement (26 microsecondes) entre l'intervalle de la température de départ et l'intervalle qui suit, puis la variation de l'erreur se stabilise pour ne varier qu'avec une valeur de 5 microsecondes. Dans [29], on a montré que la variation de la fréquence de l'oscillateur en quartz par rapport à l'augmentation de la température est une fonction linéaire. Dans [30], on a montré que la fréquence de l'oscillateur en quartz varie selon l'âge de ce dernier et sa température. Cette variation peut soit augmenter ou diminuer la fréquence de l'oscillateur en quartz. Ce qui explique les valeurs obtenues dans le graphe 4.6. Remarquons que la première variation de fréquence est importante par rapport aux autres variations. Cela s'explique par le nombre d'observations obtenues dans cet intervalle de température qui est petit par rapport aux autres intervalles car, contrairement aux études faites dans [29] et [30], ici on n'a pas de contrôle sur le nombre d'observations par rapport à la température.

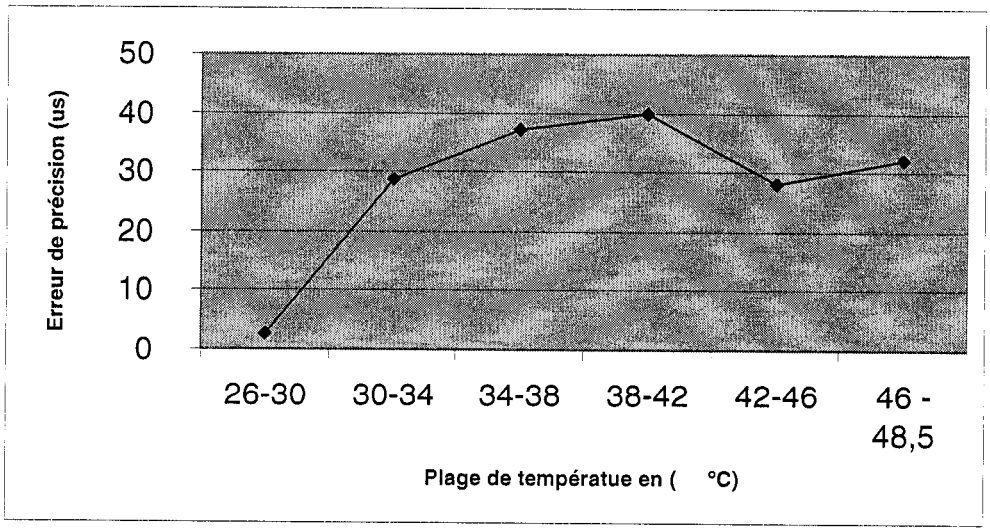

FIGURE 4.7 : Erreur de précision de l'horloge par intervalle de température (avec un degré de certitude de 99%).

4.1.2 Variation de la charge de travail du microprocesseur

L'idée ici est de voir l'effet sur l'horloge du microprocesseur causé par la variation de la charge de travail du système. La variation de cette charge entraîne une variation de la température du microprocesseur. Bien que le facteur de la température ait été étudié dans l'expérience précédente, ici la différence vient du fait que la variation de la température est non artificielle, étant due à la variation de la charge du microprocesseur.

Un ordinateur biprocesseur doté de deux Pentiums II cadencé à 350 MHz a été utilisé pour cette expérience. Trois modes de test ont été effectués :

1) Sans charge
2) Charge Moyenne
3) Charge maximale

Dans le premier mode, aucun programme ne s'exécute à part le démon qui récolte les données tracées par le noyau. Dans le deuxième mode, un script archivant le code source du noyau Linux version 2.4.26 tourne en parallèle avec le démon. Dans le troisième mode, plusieurs instances d'un programme réservant des grands tableaux à deux dimensions, de type réel, et effectuant des calculs sur ces derniers, viennent s'ajouter au deuxième mode. L'effet de ce programme est d'obliger le système à faire beaucoup de permutations en mémoire, ce qui rend le système complètement saturé.

TABLEAU 4.3 : Sommaire des statistiques de la fréquence du microprocesseur en fonction de la charge de travail.

		Aucune charge	Charge moyenne	Charge maximale
ECART TYPE	σ (cycle horloge)	149	331	426
MOYENNE	$\overline{X}$ (cycle horloge)	350 797 133	350 797 332	350 797 453
TEMPERATURE	(°C)	34 – 35	34 – 35	34 – 35
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 3{,}1\ \sigma$	$\overline{X} \pm 2{,}8\ \sigma$	$\overline{X} \pm 3\ \sigma$
	ERREUR DE PRECISION (cycle horloge)	462	926	1 279
	ERREUR DE PRECISION (µs)	1,318	2,641	3,645

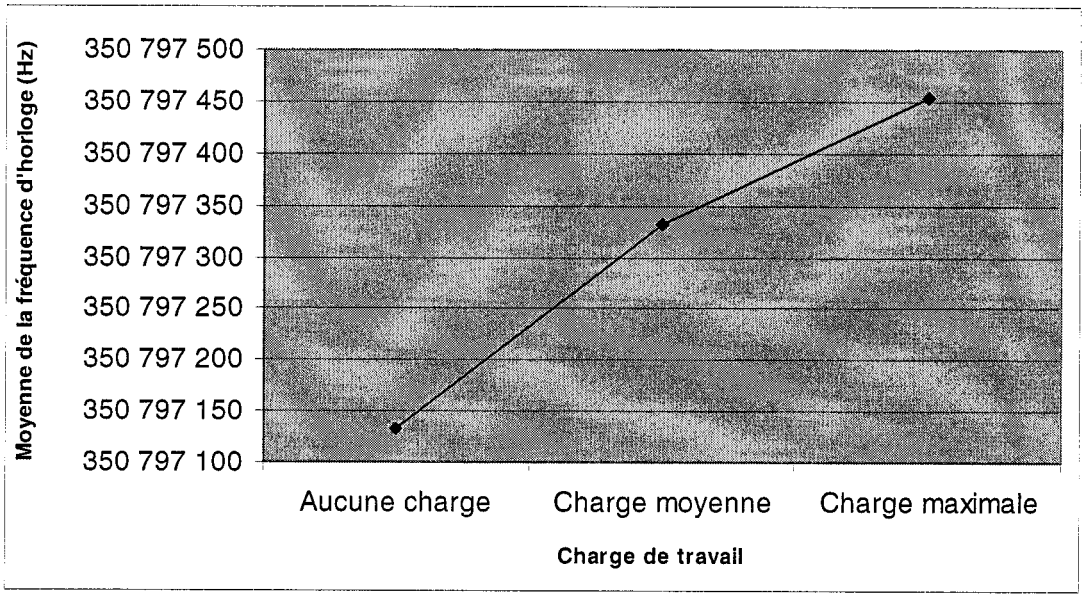

FIGURE 4.8 : La moyenne de fréquence du microprocesseur par charge de travail.

La variation de la charge de travail du microprocesseur cause une augmentation de l'erreur de précision d'une microseconde. Par contre la fréquence du microprocesseur ne

diminue pas. Comme la température du microprocesseur est restée stable, la variation dans l'erreur de précision est vraisemblablement causée par la latence des interruptions.

4.1.3 Variation d'architecture et de génération du microprocesseur

Il est temps maintenant de voir l'impact de l'architecture et de la génération du microprocesseur sur sa fréquence d'horloge. Les architectures étudiées sont i386 (Intel 32 bits et produits compatibles de AMD et VIA) et Itanium (Intel 64 bits).

Architecture i386

Constructeur AMD

Les ordinateurs qui ont servi pour ces tests sont au nombre de 8. Ils sont munis d'une carte mère ASUS A7A et de 512 Mo de DDRAM. Cinq ordinateurs sont munis d'un microprocesseur AMD 1,3 GHz et les autres ont un AMD 1,5 GHz. Le tableau 4.4 donne le sommaire des statistiques des tests.

Tableau 4.4 : Sommaire des statistiques de la fréquence d'horloge du microprocesseur AMD.

		Ordinateur 1	Ordinateur 2	Ordinateur 3	Ordinateur 4
ECART TYPE	σ (cycle horloge)	193	157	294	178
MOYENNE	$\overline{X}$ (cycle horloge)	1 343 172 580	1 343 175 836	1 343 134 661	1 343 175 117
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 3{,}42\,\sigma$	$\overline{X} \pm 4{,}06\,\sigma$	$\overline{X} \pm 3{,}21\,\sigma$	$\overline{X} \pm 3{,}16\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	661	637	943	563
	ERREUR DE PRECISION (µs)	0,492	0,475	0,702	0,419

TABLEAU 4.4 (SUITE)

		Ordinateur 5	Ordinateur 6	Ordinateur 7	Ordinateur 8
ECART TYPE	σ (cycle horloge)	407	478	362	359
MOYENNE	$\overline{X}$ (cycle horloge)	1 343 174 308	1 533 362 884	1 544 642 975	1 544 665 817
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}19\,\sigma$	$\overline{X} \pm 1{,}95\,\sigma$	$\overline{X} \pm 2{,}17\,\sigma$	$\overline{X} \pm 2{,}14\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	891	932	788	769
	ERREUR DE PRECISION (µs)	0,664	0,608	0,510	0,498

On constate qu'au degré de certitude 99 %, la moyenne de l'erreur de précision de la majorité des processeurs testés est égale à 0,5 microseconde en moyenne.

Constructeur Intel

Les ordinateurs qui ont servi pour les tests sont munis d'un microprocesseur Intel Pentium 4 cadencé à 2.4 GHz (carte mère Intel D865PE et 512 Mo DDRAM) et de Pentium 2 cadencé à 266 Mhz. Les tableaux 4.6 et 4-7 donnent le sommaire des statistiques des tests.

Pentium 4

TABLEAU 4.5 : Sommaire des statistiques de la fréquence du microprocesseur Intel Pentium 4 GHz.

		Ordinateur 1	Ordinateur 2	Ordinateur 3	Ordinateur 4
ECART TYPE	σ (cycle horloge)	2 501	2 214	2 676	2 698
MOYENNE	$\overline{X}$ (cycle horloge)	2 393 902 454	2 393 896 497	2 393 925 792	2 393 882 326
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}94\,\sigma$	$\overline{X} \pm 2{,}55\,\sigma$	$\overline{X} \pm 1{,}77\,\sigma$	$\overline{X} \pm 1{,}65\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 844	5 643	4 733	4 455
	ERREUR DE PRECISION (µs)	2,023	2,357	1,977	1,861

TABLEAU 4.5 (suite).

		Ordinateur 5	Ordinateur 6	Ordinateur 7	Ordinateur 8
ECART TYPE	σ (cycle horloge)	1 797	2 038	2 471	2 686
MOYENNE	$\overline{X}$ (cycle horloge)	2 393 889 589	2 393 857 038	2 393 886 405	2 393 882 430
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}67\,\sigma$	$\overline{X} \pm 2{,}50\,\sigma$	$\overline{X} \pm 2{,}15\,\sigma$	$\overline{X} \pm 1{,}69\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 806	5 094	5 307	4 551
	ERREUR DE PRECISION (µs)	2,008	2,128	2,217	1,901

Bien que le Pentium 4 2.4 GHz soit plus puissant que l'AMD 1,5 GHz, son erreur de précision est quatre fois supérieure (en moyenne 2 microsecondes). Cela s'explique par la grande fréquence d'horloge du Pentium. Il faudra tester des processeurs AMD cadencés à peu près à 2600 MHz afin de pouvoir statuer sur la qualité de l'horloge des deux constructeurs.

Pentium 2

TABLEAU 4.6 : Sommaire des statistiques de la fréquence du microprocesseur Intel Pentium II 266 MHz.

		Ordinateur 1	Ordinateur 2	Ordinateur 3	Ordinateur 4
ECART TYPE	σ (cycle horloge)	63	89	100	159
MOYENNE	$\overline{X}$ (cycle horloge)	266 317 841	266 314 530	266 314 969	266 318 621
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 4{,}44\,\sigma$	$\overline{X} \pm 3{,}42\,\sigma$	$\overline{X} \pm 3{,}1\,\sigma$	$\overline{X} \pm 2{,}58\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	282	303	311	411
	ERREUR DE PRECISION (µs)	1,057	1,137	1,168	1,544

TABLEAU 4.6 : (suite).

		Ordinateur 5	Ordinateur 6	Ordinateur 7
ECART TYPE	σ (cycle horloge)	75	75	107
MOYENNE	$\overline{X}$ (cycle horloge)	266 313 279	266 316 786	266 312 792
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 3{,}79\,\sigma$	$\overline{X} \pm 2{,}8\,\sigma$	$\overline{X} \pm 3{,}14\,\sigma$
	ERREUR DE PRECISION (cycleshorloge)	286	926	336
	ERREUR DE PRECISION (µs)	1,074	2,641	1,263

Avec 99% comme degré de certitude, l'erreur de précision en moyenne est égale à 1,5 microseconde, ce qui est moins précis que les processeurs AMD étudiés.

VIA

Pour ce constructeur de microprocesseur Intel32, on a eu à notre disposition qu'un seul ordinateur pour effectuer le test. Il est constitué d'une carte mère VIA EPIA ME6000, d'un microprocesseur VIA cadencé à 599 Mhz et de 512 Mo de DDRam.

TABLEAU 4.7 : Sommaire des statistiques de la fréquence du microprocesseur VIA

		Ordinateur 1
ECART TYPE	σ (cycle horloge)	637,397
MOYENNE	$\overline{X}$ (cycle horloge)	599 924 976
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}522\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	970,309
	ERREUR DE PRECISION (µs)	1,617

Itanium

En principe il faut tester plusieurs microprocesseurs. Mais on a eu à notre disposition qu'un seul serveur Intel SR870BH2 biprocesseur doté de deux microprocesseurs Itanium 1.4 GHz pour effectuer l'expérience :

TABLEAU 4.8 : Sommaire des statistiques de la fréquence du microprocesseur Itanium.

		Ordinateur 1
ECART TYPE σ (cycle horloge)		1 703
MOYENNE $\overline{X}$ (cycle horloge)		1 396 283 528
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	X ± 2,118 σ
	ERREUR DE PRECISION (cycle horloge)	3 608
	ERREUR DE PRECISION (µs)	2,584

Bien que sa fréquence soit la moitié de celle du Pentium 4 déjà testé, son erreur de précision est le double. Est-ce que cela est dû à la nouvelle architecture d'Intel ?

4.1.4 Stabilité et déviation

Une horloge stable implique une horloge fiable, car la variation de sa fréquence d'horloge au cours du temps serait toujours de la même grandeur. Le graphe qui suit montre la variance de la racine d'Allan pour 8 microprocesseurs AMD. On voit très bien que ces derniers convergent vers un état stable.

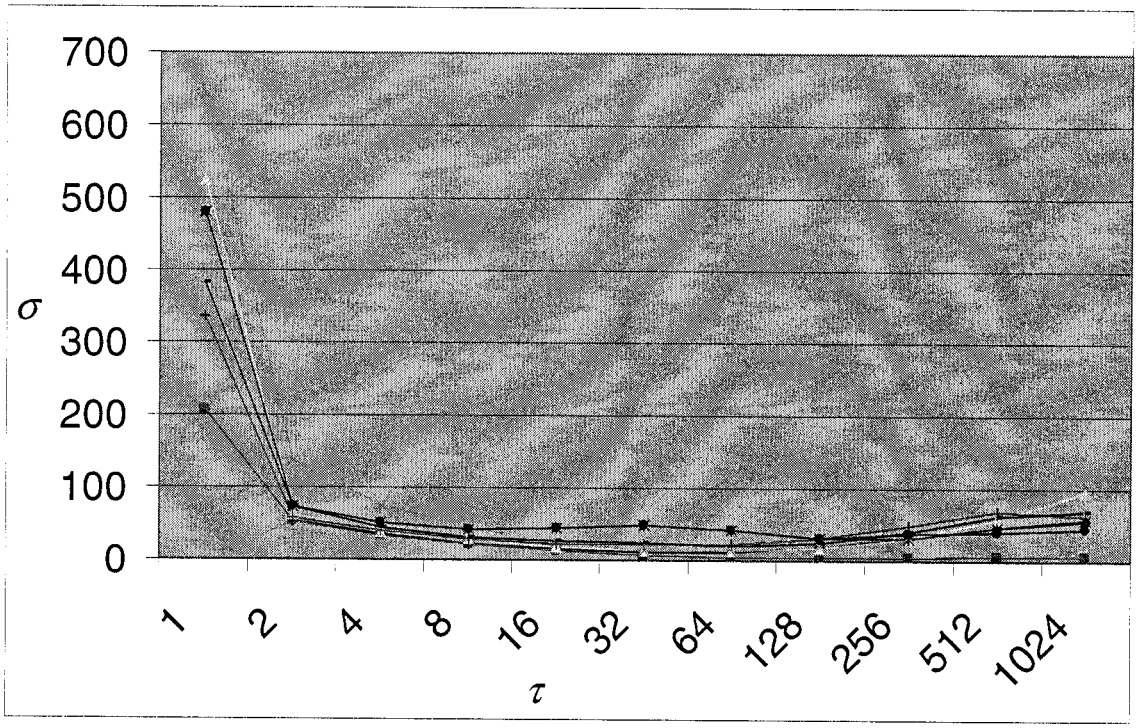

FIGURE 4.9 : Variance de la racine d'Allan pour 8 processeurs AMD

L'horloge des microprocesseurs AMD testés est caractérisée par une erreur moyenne de précision égale à 0.5 microseconde par seconde et par une bonne stabilité. Ainsi, la déviation peut être estimée dans le pire des cas à une seconde pour chaque 2 000 000 secondes ou 23.14 jours. Pour discuter la déviation sous un autre volet, les observations d'une expérience de 4 jours seront groupées par 100, pour amortir l'erreur due à la latence d'interruption, puis leurs valeurs moyennes seront calculées afin qu'on puisse constater leurs variations.

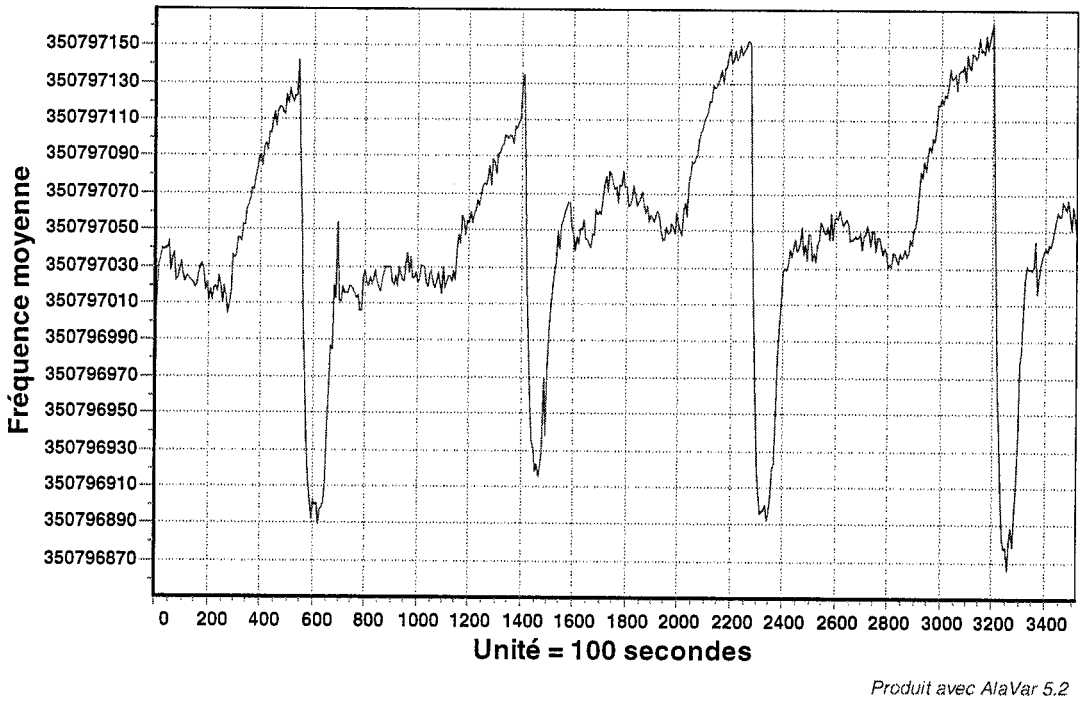

FIGURE 4.10 : Fréquence moyenne (pour 100 observations) de l'horloge du microprocesseur sur une durée de 4 jours.

Dans la figure 4.10 on constate deux types de patrons de la fréquence moyenne d'horloge du microprocesseur qui se répètent chaque 24 heures. Le premier patron correspond au fonctionnement normal de l'horloge du microprocesseur. Par contre, dans le deuxième patron, on remarque une variation anormale de la fréquence d'horloge du microprocesseur. La fréquence d'horloge du microprocesseur commence à monter à partir de 23h00 pour atteindre son sommet le plus haut vers 06h24. Puis, elle commence à chuter pour atteindre son sommet le plus bas vers 08h30. Par la suite, la fréquence

d'horloge du microprocesseur augmente jusqu'à ce qu'elle atteigne son rythme normal vers 10h00.

La fréquence d'horloge du microprocesseur varie au maximum par ± 100 cycles horloge (± 285 nanosecondes). Cette variation est cyclique et parvient aux mêmes heures, ce qui implique que quelque chose est la cause de cette variation. Pour comprendre ce phénomène, la même expérience a été refaite en débranchant l'antenne du récepteur GPS, afin de déterminer si les satellites sont la cause de ce phénomène. Les résultats obtenus sont pareils à ceux indiqués dans la figure 4.9, à part la variation de la fréquence d'horloge du microprocesseur qui a augmenté par le fait de l'imprécision du signal PPS. Ce qui reste à vérifier est l'impact de la variation de l'alimentation électrique sur l'horloge du microprocesseur en utilisant une source de tension ajustable.

Pour conclure, le registre compteur de cycles d'horloge peut être utilisé comme une source de temps à haute résolution et précision car la plupart des processeurs testés ont été caractérisé par une erreur moyenne de précision égale à une microseconde par seconde.

4.2 Réseau Ethernet

Pour ce type de réseau, deux phénomènes seront étudiés. Le premier est la variabilité du temps de transmission. Le deuxième est la variabilité du temps de diffusion. La figure 4.11 illustre l'environnement expérimental pour l'étude de ces deux phénomènes.

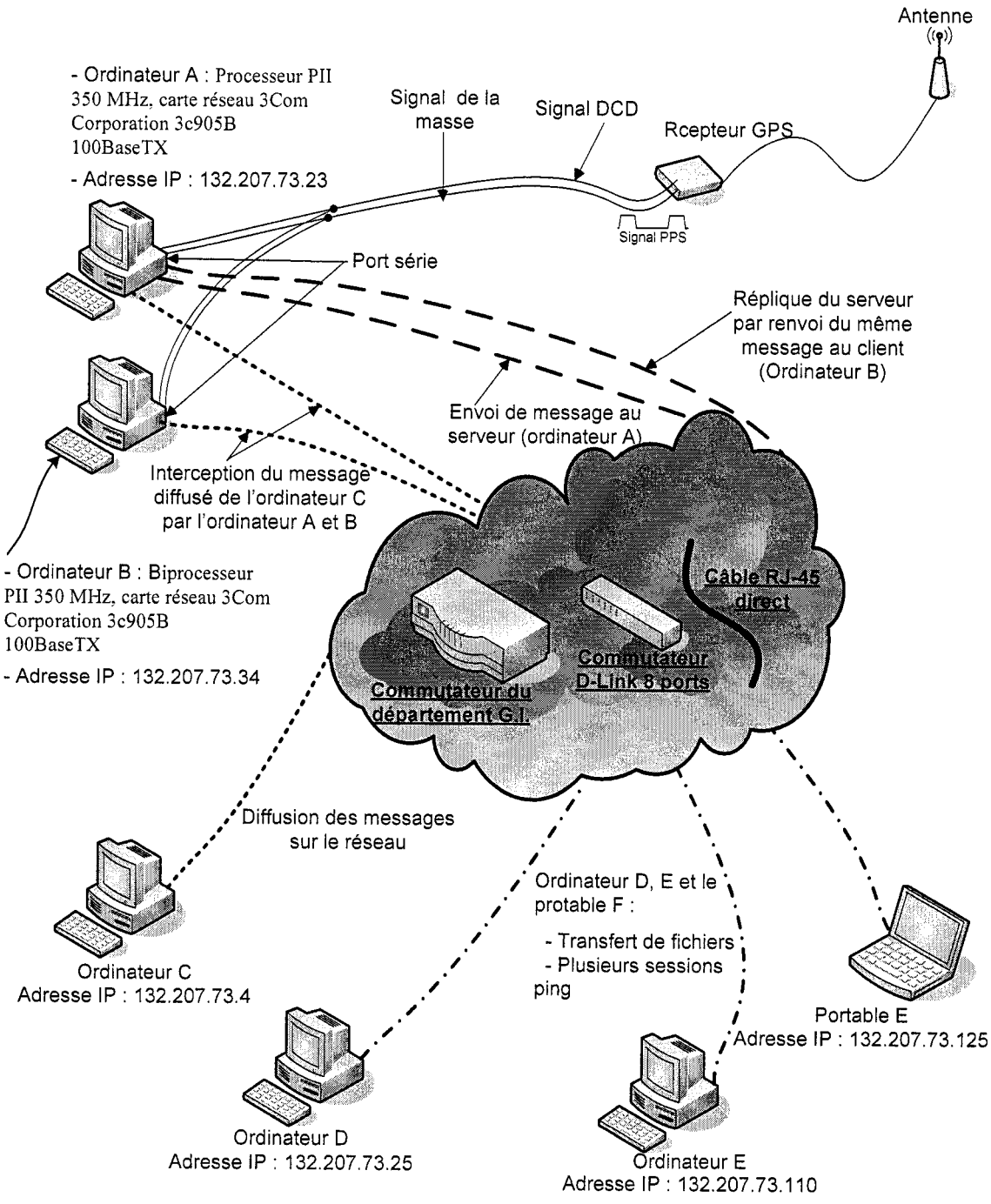

FIGURE 4.11 : Environnement expérimental pour les tests réseaux.

4.2.1 Temps de transmission

Un programme client et un programme serveur s'échangent des paquets d'information. Ces paquets sont modifiés au niveau du code source du pilote de la carte réseau pour ajouter l'heure d'arrivée ou de départ des paquets. En agissant au niveau le plus bas, le bruit qui peut être causé par le système est réduit. L'important est le temps d'aller et de retour des messages entre les deux programmes. Une référence de temps commune pour les deux programmes est donc requise. Pour cela, le signal PPS du récepteur GPS sera délivré aux deux ordinateurs sur lesquels les programmes tournent. Le signal PPS est délivré à travers le même câble avec deux connecteurs séries aux deux ordinateurs sur lesquels le programme client et serveur tournent.

L'architecture et la charge du trafic du réseau ont un impact direct sur le temps de transmission des messages. Ainsi, deux types de tests ont été effectués. Le premier type consiste à varier l'équipement qui interconnecte les ordinateurs (liaison directe avec un câble réseau croisé, réseau local du laboratoire qui est un réseau virtuel dans le commutateur du département, et finalement un commutateur pour 8 postes D-Link 10/100 Fast Ethernet). Le deuxième type de test consiste à faire varier la charge du réseau (sans charge, charge moyenne et charge maximale).

Les tests ont été effectués en branchant les ordinateurs sur le même segment réseau. A la fin de chaque expérience, un programme d'analyse est lancé afin de calculer la différence entre le temps aller et retour pour chaque observation du test. Avec ces différences, la moyenne et l'écart type sont calculés afin de déterminer la différence des

horloges des nœuds dans un réseau Ethernet qu'on peut avoir, en appliquant une variante de l'algorithme de Cristian [4] pour leurs synchronisation.

Le tableau 4.11 donne le sommaire des statistiques du temps aller et retour pour des tests effectuées en utilisant le commutateur D-Link.

TABLEAU 4.9 : Sommaire des statistiques du temps aller et retour en (μs) dans un réseau Ethernet.

		Moyenne	Écart type	Médiane	Minimum	Maximum	Étendue
100 Observations	Temps aller	120,86	1,32	121,04	117,77	123,97	6,20
	Temps retour	110,90	1,87	111,43	103,73	117,30	13,57
500 Observations	Temps aller	120,95	1,55	120,93	116,38	129,99	13,61
	Temps retour	110,47	1,48	110,54	105,11	114,48	9,38
3600 Observations	Temps aller	121,03	1,90	121,00	113,64	173,55	59,90
	Temps retour	110,53	1,60	110,66	102,77	117,30	14,53

On peut déduire que 500 observations sont suffisantes pour nos tests. Mais pour mieux valider les résultats obtenus le nombre d'observations pour nos tests sera 3600.

On remarque aussi que le temps de transmission de l'ordinateur A vers l'ordinateur B est toujours inférieur au temps de transmission de l'ordinateur B vers l'ordinateur A, même lorsque le programme client et serveur ont été permutés dans les deux ordinateurs. Cela est dû au fait que les deux ordinateurs ne sont pas équipés avec la même carte mère et que l'ordinateur B est biprocesseur.

Le but est de déterminer la variabilité entre le temps aller et retour. Ainsi, les sommaires statistiques pour le reste des expériences porteront sur la différence entre le temps aller et retour pour chaque observation.

Variation de l'équipement d'interconnexion

TABLEAU 4.10 : Sommaire des statistiques de la différence entre le temps aller et retour dans un réseau Ethernet avec différentes interconnections.

		Câble RJ-45 direct	Réseau virtuel dans le commutateur du département	Commutateur D-LINK
ECART TYPE σ (µs)		3,22	3,271	3,023
MOYENNE (µs) $\overline{X}$		11	10,161	10,493
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}30\,\sigma$	$\overline{X} \pm 2{,}26\,\sigma$	$\overline{X} \pm 2{,}32\,\sigma$
	ERREUR DE PRECISION (µs)	7,410	7,393	7,015

Bien qu'intuitivement on puisse penser que le câble direct peut donner les meilleurs résultats, les expériences semblent indiquer que le réseau virtuel du département vient en tête de classement, avec une différence de 839 nanosecondes par rapport au câble direct.

Variation de la charge (commutateur D-Link)

TABLEAU 4.11 : Sommaire des statistiques de la différence entre le temps aller et retour dans un réseau Ethernet en faisant varier la charge.

		Sans charge	Avec Charge
ECART TYPE σ (µs)		3,023	3,606
MOYENNE $\overline{X}$ (µs)		10,494	10,805
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}32\,\sigma$	$\overline{X} \pm 2{,}13\,\sigma$
	ERREUR DE PRECISION (µs)	7,015	7,695

Ici, les nœuds D, E et F, tel qu'indiqué dans figure, 4.9 effectuent des transferts de fichiers entre eux et des sessions ping afin d'augmenter la charge du réseau. Cette charge a augmenté la différence entre le temps aller et retour avec une valeur de 311 nanosecondes. Ainsi, on déduit que la charge du réseau n'a pas un grand impact sur le temps de transmission réseau.

Il est à noter que le temps aller de la transmission d'un message, pris de la couche application du client jusqu'à la couche application du serveur est nettement différent du temps retour, pris à partir de la couche application du serveur jusqu'à la couche application du client. Pour une expérience, la valeur moyenne du temps aller obtenue est égale à 112.109 microsecondes et la valeur moyenne du temps retour obtenue est égale à 72.992 millisecondes.

En résumé, en appliquant l'algorithme de Cristian pour la synchronisation des nœuds dans un réseau Ethernet 100 Mb/s, on peut s'attendre à une différence égale à 10.493 ± 7.015 µs entre les horloges des différents noeuds.

4.2.2 Temps de diffusion

Pour ce test, deux programmes s'exécutent sur deux machines qui sont à l'écoute des messages diffusés par un autre programme tournant sur une troisième machine. Le même câble série en Y utilisé auparavant sera utilisé ici, pour délivrer le signal PPS du récepteur GPS aux deux premières machines, et ainsi former la référence du temps pour les deux machines afin de mesurer le temps de diffusion du message.

Le temps de diffusion est sujet à des variations selon la charge du réseau et l'architecture de ce dernier. Ainsi, le premier type de test consiste à varier l'équipement d'interconnexion et le deuxième consiste à varier la charge de trafic réseau.

A la fin de chaque expérience, un programme d'analyse est lancé afin de calculer la différence du temps de diffusion du message aux deux machines réceptrices, pour chaque observation du test. Avec ces différences, la moyenne et l'écart type sont calculés et aident à définir le degré de synchronisation des horloges dans un réseau Ethernet en utilisant la diffusion de message.

En faisant varier le nombre d'observations on conclut que 3600 observations sont suffisantes pour le reste des expériences.

TABLEAU 4.12 : Sommaire des statistiques de la différence du temps de diffusion dans un réseau Ethernet.

	100 observations	500 observations	3600 observations
Moyenne (µs)	2,757	2,630	2,636
Écart type (µs)	0,947	1,170	1,120
Médiane (µs)	2,721	2,548	2,588
Minimum (µs)	0,776	0,043	0,011
Maximum (µs)	5,147	10,612	11,771
Étendue (µs)	4,371	10,569	11,761

Variation de l'équipement d'interconnexion

D'après le tableau 4.13 la valeur moyenne de la différence du temps de diffusion du commutateur diffère légèrement de celle du réseau virtuel du département avec 1.006

microsecondes. Mais l'erreur de précision du réseau virtuel du département est meilleure que celle du commutateur D-Link.

TABLEAU 4.13 : Sommaire des statistiques de la différence du temps de diffusion dans un réseau Ethernet.

		Réseau virtuel dans le commutateur du département	Commutateur D-LINK
ECART TYPE σ (µs)		1,164	1,129
MOYENNE $\overline{X}$ (µs)		2,411	1,405
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}794\,\sigma$	$\overline{X} \pm 2{,}740\,\sigma$
	ERREUR DE PRECISION (µs)	2,216	3,094

Variation de la charge du réseau (permutateur D-Link)

Ici, les nœuds D, E et F, comme indiqué à figure 4.9, effectuent des transferts de fichiers entre eux et des sessions ping afin d'augmenter la charge du réseau. Cette charge a augmenté la différence du temps de diffusion avec une valeur de 1,2 microseconde. Ainsi, on déduit que la charge du réseau a un impact sur le temps de diffusion d'un message.

D'après les valeurs indiquées dans le tableau 4.14 la charge du trafic réseau n'a pas un impact sur la différence du temps de diffusion.

En résumé, avec la technique de diffusion de messages pour la synchronisation des horloges des nœuds, on peut s'attendre à une différence égale à 1.405 ± 3.094 µs entre les horloges des différents noeuds.

TABLEAU 4.14 : Sommaire des statistiques de la différence du temps de diffusion dans un réseau Ethernet (selon la charge)

		Aucune charge	Avec charge
ECART TYPE σ		1,129	1,120
MOYENNE $\overline{X}$		1,405	2,636
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}740\,\sigma$	$\overline{X} \pm 2{,}557\,\sigma$
	ERREUR DE PRECISION (µs)	3,094	2,864

Conclusion

Au cours de ce mémoire, les trois points mentionnés dans l'introduction pour la mise en œuvre d'une version parallèle de l'outil d'analyse de performance LTT ont été étudiés.

Le premier point concerne l'étude de la possibilité d'utiliser le registre compteur de cycles d'horloge du microprocesseur comme source de temps à haute résolution et précision, pouvant répondre aux exigences de Linux Trace Toolkit (LTT permet de tracer des événements pour lesquels l'écart de temps est de quelques nanosecondes). Les travaux existants sur la stabilité et la déviation de ce registre [21] ont été étendus afin d'obtenir des résultats beaucoup plus précis. Ceci en utilisant une référence de temps plus précise et fiable (un récepteur GPS, délivrant un signal PPS à chaque seconde avec une précision de ± 12 nanosecondes) pour l'étude du registre et en réduisant la latence des interruptions au minimum en interceptant le signal PPS du récepteur GPS au plus bas niveau du système Linux. Les facteurs qui peuvent influencer la fréquence d'horloge du microprocesseur, tels que la variation de sa température et de sa charge du travail ont été étudiés et caractérisés. Finalement, différents types de microprocesseurs (Intel i386, Intel 64 bits, AMD et VIA) ont été étudiés. Ceci nous a conduit à conclure que le compteur de cycles d'horloge du microprocesseur est caractérisé avec une bonne

stabilité et s'avère plus adéquat que l'horloge du système Linux en termes de résolution (une résolution d'un cycle, soit typiquement moins d'une nanoseconde, avec une erreur de précision de ± 1 microseconde / seconde). Il serait intéressant dans le futur d'analyser l'effet de la variation de l'alimentation électrique sur la fréquence d'horloge.

Pour corréler des traces provenant de différents nœuds, il faut convertir les valeurs lues du registre compteur de cycles d'horloge vers une base de temps commune, d'où la nécessité de déterminer la fréquence d'horloge du microprocesseur. Linux détermine cette fréquence au démarrage de l'ordinateur en se basant sur l'oscillateur du programmeur d'interruption programmable (PIT). Avec cette manière de faire, on obtient une estimation limitée par la précision de l'oscillateur du PIT. En outre, les expériences montrent qu'au démarrage de l'ordinateur le microprocesseur fonctionne à une température nominale, sa fréquence initiale varie ainsi au cours du temps jusqu'à ce que ce dernier atteigne sa température de fonctionnement normal. Ainsi, pour déterminer précisément cette fréquence il faut utiliser une source de temps externe fiable comme le signal PPS d'un récepteur GPS, et ne déterminer la fréquence du microprocesseur que lorsque le microprocesseur a atteint sa température de fonctionnement normal.

Pour le deuxième point, concernant la synchronisation des horloges dans une grappe d'ordinateurs, des solutions existent déjà. Elles sont basées sur un principe probabiliste concernant la symétrie du délai de transfert ou de diffusion des messages de synchronisation entre les nœuds à synchroniser.

Le premier type de solution est générique et inclut NTP [3], mais la synchronisation atteinte par cette dernière ne peut pas répondre aux exigences de LTT. En effet, dans un

réseau local à 100 Mbps, connecté à un serveur NTP primaire, la synchronisation est de l'ordre de 1ms. Le deuxième type de solution est spécifique à une architecture et un équipement réseau donné, comme la solution [13] pour les réseaux Myrinet qui a donné de bons résultats; dans un réseau chargé, la synchronisation est de l'ordre de 1,45 ± 1,26 µs.

Tel que vu à la section 1.4.3, la synchronisation des horloges des nœuds dans un système réparti dépend de deux facteurs. Le premier facteur est le temps pris par le système d'exploitation pour constituer le message à transmettre. Le deuxième facteur est le temps réel de transmission dans le réseau. Dans notre étude, le premier facteur a été éliminé en interceptant les messages de synchronisation des nœuds au plus bas niveau du système Linux, pour n'étudier que la variabilité du délai de transfert et de diffusion pour la technologie réseau la plus répandue, Ethernet 100 Mbps. Ceci permet de statuer sur le degré de synchronisation qu'on peut atteindre dans ce type de réseau.

Il serait souhaitable, lors de travaux subséquents, de vérifier le délai associé au système d'exploitation selon le point d'insertion (plus ou moins spécifique à une carte réseau et un protocole) dans le système de la lecture du compteur de cycle. Cette lecture est effectuée à l'envoi et à la réception des paquets. Il serait ainsi possible de trouver le meilleur compromis entre le délai et la facilité d'insertion.

Les résultats obtenus suggèrent l'utilisation de la technique de diffusion pour la synchronisation des nœuds de la grappe, car elle est 10 fois plus précise que la technique de transfert; les nœuds peuvent être synchronisés avec une précision égale à 1.405 ± 3.094 µs. L'inconvénient est le fait que la technique de diffusion ne peut être appliquée que sur des nœuds qui sont dans le même segment réseau. Dans une grappe avec

plusieurs nœuds, on risque de se trouver avec différents segments, contrairement à la technique de transmission de messages qui est d'usage générique. Dans [31], une solution à ce problème a produit de bons résultats, bien qu'elle nécessite une conception spéciale du réseau.

Finalement, il serait intéressant d'appliquer une variante de l'algorithme de synchronisation des horloges [13] de la grappe en se basant sur l'échange des messages entre les nœuds du système, puis un algorithme de synchronisation des horloges des nœuds basé sur la diffusion des messages aux nœuds de la grappe. Ceci permettrait de mettre en évidence les résultats obtenus dans ce mémoire.

Bibliographie

[1] Linux Trace Toollkit home page. [En ligne]. http://www.opersys.com/LTT (Page consultée le 10 septembre 2004)

[2] MAMMOUTH. [En ligne]. http://salle.ccs.usherbrooke.ca/ (Page consultée le 10 septembre 2004)

[3] NTP. [En ligne]. http://www.ntp.org/ (Page consultée le 10 septembre 2004)

[4] Coulouris, G., Dollimore, J., Kindberg, T. 2001. "TIME AND GLOBALE STATE" *Distributed Systems Concepts and Design.* ADDISON WESLEY. P. 385-400.

[5] Yaghmour, K. 2001. *Analyse de Performance et Caractérisation de Comportement à l'Aide d'Enregistrement d'Evénements Noyau.* Mémoire de maîtrise es sciences appliquées. École Polytechnique de Montréal, Canada.

[6] Hollingsworth, J., Miller, B. 1992. "Parallel Program Performance Metrics : A Comparaison and Validation". *In Proceedings of Supercomputing*, pp 4–13.

[7] Chandy, K., Misra, J. 1982. "Distributed computation on graphs : Shortest path algorithms". *CACM* 25. pp. 833-837.

[8] Yang, C., Miller, B. 1988. "Critical Path Analysis for the Execution of Parallel and Distributed Programs". *8th International Conference on Distributed Computing Systems.* pp 366–373.

[9] Anderson, T. E., Lazowska, E. D. 1990. "Quartz: a tool for tuning parallel program performance". *In Proceedings of ACM SIGMETRICS conference on Measurement and modeling of computer systems.*

[10] Li, T., Lebeck, A. R., Sorin, D. J. 2003. "Quantifying Instruction Criticality for Shared Memory Multiprocessors". *In proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures.*

[11] Gusella, R., Zatti, S. 1989. "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD". *IEEE Transactions on Software Engineering*. Volume: 15, pp. 847-853.

[12] Juhász, S., Charaf, H. 2004. "Parallel and distributed systems (PDS): Exploiting fast ethernet performance in multiplatform cluster environment". *Proceedings of the 2004 ACM symposium on Applied computing.*

[13] Liao, C., Martonosi, M., Clark, D. W. 1999. "Experience with an adaptive globally-synchronizing clock algorithm". *Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures.*

[14] BOVET, D. P., CESATI, M. 2002. *Understanding the Linux Kernel.* O'REILLY.

[15] RUBINI, A. 2002. *Linux Device Drivers.* O'REILLY.

[16] Intel. 1997. *Intel Architecture Software Developers Manual, Volume 3: System Programming Guide.*

[17] Rieker, M. 2002. Advanced Programmable Interrupt Controller. [En ligne]. http://osdev.berlios.de/pic.html (Page consultée le 10 septembre 2004)

[18] Mosberger, D., Eranian, S. 2002. *I64 BOOK IA-64 Linux Kernel: Design and Implementation* .Prentice Hall PTR.

[19] Johnny Appleseed G P S. 2004. The Theory and Practice of GPS. [En ligne]. http://www.ja-gps.com.au/whatisgps.html (Page consultée le 10 septembre 2004)

[20] GPS, UTC, and TAI Clocks. [En ligne]. http://www.leapsecond.com/java/gpsclock.htm (Page consultée le 10 septembre 2004)

[21] Pásztor, A., Veitch, D. 2002. "PC based precision timing without GPS". *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. Volume 30 Issue 1.

[22] MPI - The Message Passing Interface Standard. [En ligne]. http://www-unix.mcs.anl.gov/mpi/ (Page consultée le 10 septembre 2004)

[23] Smotlacha, V. 2001. "Measurement of Time Servers".

[24] DePriest, D. 2002. How Your GPS Works. [En ligne]. http://www.gpsinformation.org/dale/theory.htm (Page consultée le 10 septembre 2004)

[25] Makdissi, A. 2003. AlaVar. [En ligne]. http://perso.club-internet.fr/yazdiet/av_help/ (Page consultée le 10 septembre 2004)

[26] Allan, D. W. 2004. The Allan Variance. [En ligne]. http://www.allanstime.com/AllanVariance/ (Page consultée le 10 septembre 2004)

[27] Duntemann, J. 2000. *Linux Assembly Language Programming*. Prentice Hall PTR.

[28] William, B., Austin, W. 2000. *Advanced PC Architecture*. Addison-Wesley.

[29] Kawashima, H. Sunaga, K. 1991. "Temperature compensated crystal oscillator employing new shape GT cut quartz crystal resonator". *Proceedings of the 45th Annual Symposium on Frequency Control*. 29-31. Pages:410 - 417

[30] Karlquist, R.K. et al. 1997. « A low-profile high-performance crystal oscillator for timekeeping applications". *Proceedings of the 1997 Symposium on Frequency Control*. Pages:873 – 884

[31] Elson, J., Girod, L., Estrin, D. 2002. "Fine-grained network time synchronization using reference broadcasts". *Proceedings of the 5th symposium on Operating systems design and implementation*. Volume 36 , Issue SI. Pages: 147 – 163.

[32] Manugistics, Inc. 2001. *STATGRAPHICS*Plus*. [Logiciel]

[33] Lm-sensor-2.8.7. [Logiciel]. http://secure.netroedge.com/~lm78/index.html

Annexe I

Registre compteur de cycles d'horloge du microprocesseur

I.1 Variation de la température

TABLEAU I.1 : Sommaire des statistiques de la fréquence du microprocesseur en fonction de la température

	INTERVALLE DE TEMPERATURE (°C)					
	26-30	30-34	34-38	38-42	42-46	46 - 48,5
ECART TYPE σ (cycle horloge)	464,67	1 686,58	1 878,87	2 299,72	1 469,95	1 929,43
MOYENNE $\overline{X}$ (cycle horloge)	349 205 333	349 204 698	349 204 068	349 203 314	349 202 799	349 202 435
MEDIANE	349 205 266	349 204 677	349 204 220	349 203 454	349 202 732	349 202 596
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)	349 204 868	349 203 011	349 202 189	349 201 015	349 201 330	349 200 505
BORNE MAXIMALE $\overline{X} + \sigma$ (cycle horloge)	349 205 798	349 206 384	349 205 947	349 205 614	349 204 269	349 204 364
VARIANCE	215922,15	2844543,10	3530168,16	5288691,57	2160743,76	3722718,27
MINIMUM (cycle horloge)	349 203 556	349 188 021	349 183 010	349 187 144	349 187 832	349 185 529
MAXIMUM (cycle horloge)	349 207 264	349 221 377	349 224 004	349 219 813	349 218 094	349 218 742
ETENDUE (cycle horloge)	3 708	33 356	40 994	32 669	30 262	33 213
INTERVALLE DE CONFIANCE A ≈ 99 %	$\overline{X} \pm 2{,}08\ \sigma$	$\overline{X} \pm 5{,}94\ \sigma$	$\overline{X} \pm 6{,}89\ \sigma$	$\overline{X} \pm 6{,}05\ \sigma$	$\overline{X} \pm 6{,}68\ \sigma$	$\overline{X} \pm 5{,}78\ \sigma$
ERREUR DE PRECISION (cycle horloge) A ≈ 99 %	967	10 018	12 964	13 936	9 819	11 191
ERREUR DE PRECISION (µs) A ≈ 99 %	2,768	28,689	37,125	39,909	28,119	32,047

I.2 Variation de la charge de travail du microprocesseur

TABLEAU I.2 : Sommaire des statistiques de la fréquence du microprocesseur en fonction de la charge de travail.

		Aucune charge	Charge moyenne	Charge maximale
ECART TYPE	σ (cycle horloge)	149	331	426
MOYENNE	$\overline{X}$ (cycle horloge)	350 797 133	350 797 332	350 797 453
MEDIANE	(cycle horloge)	350 797 111	350 797 333	350 797 454
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)		350 796 984	350 797 002	350 797 027
BORNE MAXIMALE $\overline{X} + \sigma$ (cycle horloge)		350 797 282	350 797 663	350 797 880
VARIANCE		22 239,79	109 466,84	181 676,03
MAXIMUM	(cycle horloge)	350 796 096	350 795 763	350 795 739
MINIMUM	(cycle horloge)	350 798 332	350 798 684	350 799 463
ETENDUE	(cycle horloge)	2 236	2 921	3 724
TEMPERATURE	(°C)	34 – 35	34 – 35	34 – 35
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}74\ \sigma$	$\overline{X} \pm 1{,}66\ \sigma$	$\overline{X} \pm 1{,}66\ \sigma$
	ERREUR DE PRECISION (cycle horloge)	259	549	708
	ERREUR DE PRECISION (µs)	0,740	1,566	2,017
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}1\ \sigma$	$\overline{X} \pm 2\ \sigma$	$\overline{X} \pm 2{,}1\ \sigma$
	ERREUR DE PRECISION (cycle horloge)	313	662	895
	ERREUR DE PRECISION (µs)	0,893	1,886	2,552
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 3{,}1\ \sigma$	$\overline{X} \pm 2{,}8\ \sigma$	$\overline{X} \pm 3\ \sigma$
	ERREUR DE PRECISION (cycle horloge)	462	926	1 279
	ERREUR DE PRECISION (µs)	1,318	2,641	3,645

I.3 Variation d'architecture et de génération du microprocesseur

Architecture i386

Constructeur AMD

TABLEAU I.3 : Sommaire des statistiques de la fréquence d'horloge du microprocesseur AMD.

		Ordinateur 1	Ordinateur 2	Ordinateur 3	Ordinateur 4
ECART TYPE σ (cycle horloge)		193	157	294	178
MOYENNE $\overline{X}$ (cycle horloge)		1 343 172 580	1 343 175 836	1 343 134 661	1 343 175 117
MEDIANE (cycle horloge)		1 343 172 558	1 343 175 840	1 343 134 600	1 343 175 078
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)		1 343 172 386	1 343 172 386	1 343 175 679	1 343 134 367
BORNE MAXIMALE $\overline{X} + \sigma$ V		1 343 172 773	1 343 172 773	1 343 175 993	1 343 134 955
VARIANCE		37 313,70	24 647,09	86 375,62	31 691,13
MAXIMUM (cycle horloge)		1 343 171 280	1 343 174 238	1 343 133 640	1 343 174 118
MINIMUM (cycle horloge)		1 343 173 598	1 343 176 598	1 343 135 800	1 343 177 971
ETENDUE (cycle horloge)		2 318	2 360	2 160	3 853
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}54\,\sigma$	$\overline{X} \pm 0{,}56\,\sigma$	$\overline{X} \pm 1{,}76\,\sigma$	$\overline{X} \pm 1{,}36\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	297	88	517	242
	ERREUR DE PRECISION (µs)	0,221	0,065	0,385	0,180
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}78\,\sigma$	$\overline{X} \pm 3{,}51\,\sigma$	$\overline{X} \pm 2{,}81\,\sigma$	$\overline{X} \pm 2{,}06\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	537	551	826	367
	ERREUR DE PRECISION (µs)	0,400	0,410	0,615	0,273
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 3{,}42\,\sigma$	$\overline{X} \pm 4{,}06\,\sigma$	$\overline{X} \pm 3{,}21\,\sigma$	$\overline{X} \pm 3{,}16\,\sigma$

	ERREUR DE PRECISION (cycles horloge)	661	637	943	563
	ERREUR DE PRECISION (µs)	0,492	0,475	0,702	0,419

TABLEAU I.3 (SUITE)

		Ordinateur 5	Ordinateur 6	Ordinateur 7	Ordinateur 8
ECART TYPE	σ (cycle horloge)	407	478	362	359
MOYENNE	$\overline{X}$ (cycle horloge)	1 343 174 308	1 533 362 884	1 544 642 975	1 544 665 817
MEDIANE	(cycle horloge)	1 343 174 278	1 533 362 896	1 544 642 924	1 544 665 786
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)		1 343 172 386	1 533 362 405	1 533 362 405	1 544 642 613
BORNE MAXIMALE $\overline{X} + \sigma$ (cycle horloge)		1 343 172 773	1 533 363 362	1 533 363 362	1 544 643 337
VARIANCE		165 637,43	228 652,40	131 107,86	129 173,65
MAXIMUM	(cycle horloge)	1 343 172 542	1 533 354 856	1 544 640 810	1 544 663 663
MINIMUM	(cycle horloge)	1 343 175 280	1 533 370 812	1 544 643 936	1 544 666 704
ETENDUE	(cycle horloge)	2 738	15 956	3 126	3 041
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}89\,\sigma$	$\overline{X} \pm 1.71\,\sigma$	$\overline{X} \pm 1{,}885\,\sigma$	$\overline{X} \pm 1{,}881\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	769	818	683	676
	ERREUR DE PRECISION (µs)	0,573	0,533	0,442	0,438
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2\,\sigma$	$\overline{X} \pm 1{,}81\,\sigma$	$\overline{X} \pm 1{,}92\,\sigma$	$\overline{X} \pm 2{,}00\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	814	865	697	719
	ERREUR DE PRECISION (µs)	0,606	0,564	0,451	0,466
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}19\,\sigma$	$\overline{X} \pm 1{,}95\,\sigma$	$\overline{X} \pm 2{,}17\,\sigma$	$\overline{X} \pm 2{,}14\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	891	932	788	769
	ERREUR DE PRECISION (µs)	0,664	0,608	0,510	0,498

Constructeur Intel

Pentium 4

TABLEAU I.4 : Sommaire des statistiques de la fréquence du microprocesseur Intel Pentium 4 GHz.

		Ordinateur 1	Ordinateur 2	Ordinateur 3	Ordinateur 4
ECART TYPE	σ (cycle horloge)	2 501	2 214	2 676	2 698
MOYENNE	$\overline{X}$ (cycle horloge)	2 393 902 454	2 393 896 497	2 393 925 792	2 393 882 326
MEDIANE	(cycle horloge)	2 393 902 076	2 393 895 204	2 393 927 968	2 393 884 764
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)		2 393 899 953	2 393 899 953	2 393 894 283	2 393 923 117
BORNE MAXIMALE $\overline{X} + \sigma$ (cycle horloge)		2 393 904 955	2 393 904 955	2 393 898 711	2 393 928 468
VARIANCE		6 254 006,86	4 902 509,13	7 159 041,71	7 276 667,76
MAXIMUM	(cycle horloge)	2 393 900 248	2 393 889 914	2 393 920 986	2 393 877 806
MINIMUM	(cycle horloge)	2 393 908 712	2 393 902 232	2 393 928 204	2 393 885 022
ETENDUE	(cycle horloge)	8 464	12 318	7 218	7 216
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}92\,\sigma$	$\overline{X} \pm 1{,}75\,\sigma$	$\overline{X} \pm 1{,}126\,\sigma$	$\overline{X} \pm 1{,}02\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 807	3 880	3 013	2 745
	ERREUR DE PRECISION (µs)	2,008	1,621	1,259	1,147
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}92\,\sigma$	$\overline{X} \pm 2{,}51\,\sigma$	$\overline{X} \pm 1{,}73\,\sigma$	$\overline{X} \pm 1{,}64\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 809	5 559	4 640	4 428
	ERREUR DE PRECISION (µs)	2,009	2,322	1,938	1,850
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}94\,\sigma$	$\overline{X} \pm 2{,}55\,\sigma$	$\overline{X} \pm 1{,}77\,\sigma$	$\overline{X} \pm 1{,}65\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 844	5 643	4 733	4 455

	ERREUR DE PRECISION (µs)	2,023	2,357	1,977	1,861

TABLEAU I.4 (suite).

		Ordinateur 5	Ordinateur 6	Ordinateur 7	Ordinateur 8
ECART TYPE	σ (cycle horloge)	1 797	2 038	2 471	2 686
MOYENNE	$\overline{X}$ (cycle horloge)	2 393 889 589	2 393 857 038	2 393 886 405	2 393 882 430
MEDIANE	(cycle horloge)	2 393 889 982	2 393 857 152	2 393 884 800	2 393 884 776
BORNE MINIMALE	$\overline{X} - \sigma$ (cycle horloge)	2 393 887 792	2 393 855 000	2 393 883 934	2 393 879 743
BORNE MAXIMALE	$\overline{X} + \sigma$ (cycle horloge)	2 393 891 385	2 393 859 076	2 393 888 876	2 393 885 116
VARIANCE		3 227 465,55	4 153 435,22	6 104 314,72	7 216 962,44
MAXIMUM	(cycle horloge)	2 393 884 466	2 393 851 544	2 393 884 486	2 393 877 790
MINIMUM	(cycle horloge)	2 393 891 886	2 393 859 398	2 393 896 894	2 393 885 080
ETENDU	(cycle horloge)	7 420	7 854	12 408	7 290
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}66\,\sigma$	$\overline{X} \pm 2{,}47\,\sigma$	$\overline{X} \pm 1{,}448\,\sigma$	$\overline{X} \pm 1{,}05\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 787	5 037	3 028	2 832
	ERREUR DE PRECISION (µs)	2,000	2,104	1,494	1,183
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}67\,\sigma$	$\overline{X} \pm 2{,}49\,\sigma$	$\overline{X} \pm 1{,}458\,\sigma$	$\overline{X} \pm 1{,}68\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 790	5 070	3 602	4 524
	ERREUR DE PRECISION (µs)	2,001	2,118	1,505	1,890
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}67\,\sigma$	$\overline{X} \pm 2{,}50\,\sigma$	$\overline{X} \pm 2{,}15\,\sigma$	$\overline{X} \pm 1{,}69\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	4 806	5 094	5 307	4 551
	ERREUR DE PRECISION (µs)	2,008	2,128	2,217	1,901

Pentium 2

TABLEAU I.5 : Sommaire des statistiques de la fréquence du microprocesseur Intel Pentium 2 266 MHz.

		Ordinateur 1	Ordinateur 2	Ordinateur 3	Ordinateur 4
ECART TYPE	σ (cycle horloge)	63	89	100	159
MOYENNE	$\overline{X}$ (cycle horloge)	266 317 841	266 314 530	266 314 969	266 318 621
MEDIANE	(cycle horloge)	266 317 840	266 314 504	266 314 944	266 318 602
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)		266 317 778	266 314 441	266 314 869	266 318 462
BORNE MAXIMALE $\overline{X} + \sigma$ (cycle horloge)		266 317 904	266 314 619	266 315 070	266 318 781
VARIANCE		4 019,68	7 837,56	10 065,35	25 386,73
MAXIMUM	(cycle horloge)	266 317 524	266 314 172	266 314 172	266 318 001
MINIMUM	(cycle horloge)	266 318 149	266 315 032	266 315 702	266 319 291
ETENDUE	(cycle horloge)	625	860	1 530	1 290
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}42\,\sigma$	$\overline{X} \pm 1{,}74\,\sigma$	$\overline{X} \pm 1{,}42\,\sigma$	$\overline{X} \pm 1{,}7\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	90	154	142	271
	ERREUR DE PRECISION (µs)	0,338	0,578	0,535	1,017
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}88\,\sigma$	$\overline{X} \pm 2{,}28\,\sigma$	$\overline{X} \pm 2{,}1\,\sigma$	$\overline{X} \pm 2{,}16\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	183	202	211	344
	ERREUR DE PRECISION (µs)	0,686	0,758	0,791	1,292
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 4{,}44\,\sigma$	$\overline{X} \pm 3{,}42\,\sigma$	$\overline{X} \pm 3{,}1\,\sigma$	$\overline{X} \pm 2{,}58\,\sigma$
	ERREUR DE PRECISION (cycles horloge)	282	303	311	411
	ERREUR DE PRECISION (µs)	1,057	1,137	1,168	1,544

TABLEAU I.5 : (suite).

		Ordinateur 5	Ordinateur 6	Ordinateur 7
ECART TYPE	σ (cycle horloge)	75	75	107
MOYENNE	$\overline{X}$ (cycle horloge)	266 313 279	266 316 786	266 312 792
MEDIANE	(cycle horloge)	266 313 288	266 316 790	266 312 816
BORNE MINIMALE	$\overline{X} - \sigma$ (cycle horloge)	266 317 778	266 313 203	266 316 711
BORNE MAXIMALE	$\overline{X} + \sigma$ (cycle horloge)	266 317 904	266 313 354	266 316 861
VARIANCE		5 700,20	5 603,00	11 473,39
MAXIMUM	(cycle horloge)	266 312 764	266 316 208	266 312 176
MINIMUM	(cycle horloge)	266 313 836	266 317 380	266 313 288
ETENDU	(cycle horloge)	1 072	1 172	1 112
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}42\,\sigma$	$\overline{X} \pm 1{,}57\,\sigma$	$\overline{X} \pm 1{,}8\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	107	118	193
	ERREUR DE PRECISION (µs)	0,403	0,441	0,724
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}56\,\sigma$	$\overline{X} \pm 2{,}59\,\sigma$	$\overline{X} \pm 2{,}48\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	193	194	266
	ERREUR DE PRECISION (µs)	0,726	0,728	0,997
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 3{,}79\,\sigma$	$\overline{X} \pm 2{,}8\,\sigma$	$\overline{X} \pm 3{,}14\,\sigma$
	ERREUR DE PRECISION (cycleshorloge)	286	926	336
	ERREUR DE PRECISION (µs)	1,074	2,641	1,263

VIA

TABLEAU I.6 : Sommaire des statistiques de la fréquence du microprocesseur VIA

		Ordinateur 1
ECART TYPE	σ (cycle horloge)	637,397
MOYENNE	$\overline{X}$ (cycle horloge)	599 924 976
MEDIANE	(cycle horloge)	599 924 556
BORNE MINIMALE $\overline{X}$ - σ (cycle horloge)		599 924 339
BORNE MAXIMALE $\overline{X}$ + σ (cycle horloge)		599 925 614
VARIANCE		406 275,338
MAXIMUM	(cycle horloge)	599 924 316
MINIMUM	(cycle horloge)	599 925 952
ETENDUE	(cycle horloge)	1 636
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}522\ \sigma$
	ERREUR DE PRECISION (cycle horloge)	970,278
	ERREUR DE PRECISION (µs)	1,617
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}522\,\sigma$
	ERREUR DE PRECISION (cycle horloge)	970,309
	ERREUR DE PRECISION (µs)	1,617

Itanium

TABLEAU I.7 : Sommaire des statistiques de la fréquence du microprocesseur Itanium.

		Ordinateur 1
ECART TYPE σ (cycle horloge)		1 703
MOYENNE $\overline{X}$ (cycle horloge)		1 396 283 528
MEDIANE (cycle horloge)		1 396 284 234
BORNE MINIMALE $\overline{X} - \sigma$ (cycle horloge)		1 089 131 539
BORNE MAXIMALE $\overline{X} + \sigma$ (cycle horloge)		1 089 132 577
VARIANCE		2 899 204,38
MAXIMUM (cycle horloge)		1 396 277 651
MINIMUM (cycle horloge)		1 396 288 537
ETENDUE (cycle horloge)		10 886
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	X ± 1,444 Sig
	ERREUR DE PRECISION (cycle horloge)	2 460
	ERREUR DE PRECISION (µs)	1,762
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	X ± 1,622 Sig
	ERREUR DE PRECISION (cycle horloge)	2 762
	ERREUR DE PRECISION (µs)	1,978
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	X ± 2,118 Sig
	ERREUR DE PRECISION (cycle horloge)	3 608
	ERREUR DE PRECISION (µs)	2,584

Annexe II

Réseau Ethernet

II.1 Temps de transmission

Variation de l'équipement d'interconnexion

TABLEAU II.1 : Sommaire des statistiques de la différence entre le temps aller et retour dans un réseau Ethernet avec différentes interconnections.

		Câble RJ-45 direct	Réseau virtuel dans le commutateur du département	Commutateur D-LINK
ECART TYPE σ (µs)		3,22	3,271	3,023
MOYENNE (µs) $\overline{X}$		11	10,161	10,493
MEDIANE (µs)		11	9,982	10,420
BORNE MINIMALE $\overline{X} - \sigma$ (µs)		8	7	7
BORNE MAXIMALE $\overline{X} + \sigma$ (µs)		14	13	14
VARIANCE		10,37	10,702	9,141
MINIMUM (µs)		4	0,904	0,967
MAXIMUM (µs)		76	69,025	59,947
ETENDUE (µs)		72	68	59
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}37\,\sigma$	$\overline{X} \pm 1{,}32\,\sigma$	$\overline{X} \pm 1{,}65\,\sigma$
	ERREUR DE PRECISION (µs)	4,420	4,320	4,990
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}42\,\sigma$	$\overline{X} \pm 1{,}64\,\sigma$	$\overline{X} \pm 1{,}66\,\sigma$
	ERREUR DE PRECISION (µs)	4,582	5,360	5,010
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}30\,\sigma$	$\overline{X} \pm 2{,}26\,\sigma$	$\overline{X} \pm 2{,}32\,\sigma$
	ERREUR DE PRECISION (µs)	7,410	7,393	7,015

Variation de la charge (commutateur D-Link)

TABLEAU II.2 : Sommaire des statistiques de la différence entre le temps aller et retour dans un réseau Ethernet en faisant varier la charge.

		Sans charge	Avec Charge
ECART TYPE σ (µs)		3,023	3,606
MOYENNE $\overline{X}$ (µs)		10,493	10,805
MEDIANE (µs)		10	10,595
BORNE MINIMALE $\overline{X} - \sigma$ (µs)		7	7
BORNE MAXIMALE $\overline{X} + \sigma$ (µs)		14	14
VARIANCE		9,14	13,004
MINIMUM (µs)		1	1,156
MAXIMUM (µs)		60	68,842
ETENDUE (µs)		59	68
NIVEAU DE CONFIANCE ≈ 90 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}65\,\sigma$	$\overline{X} \pm 1{,}302\,\sigma$
	ERREUR DE PRECISION (µs)	4,990	4,695
NIVEAU DE CONFIANCE ≈ 95 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}66\,\sigma$	$\overline{X} \pm 1{,}58\,\sigma$
	ERREUR DE PRECISION (µs)	5,010	5,695
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}32\,\sigma$	$\overline{X} \pm 2{,}13\,\sigma$
	ERREUR DE PRECISION (µs)	7,015	7,695

II.2 Temps de diffusion

Variation de l'équipement d'interconnexion

TABLEAU II.3 : Sommaire des statistiques de la différence du temps de diffusion dans un réseau Ethernet.

	Réseau virtuel dans le commutateur du département	Commutateur D-LINK
ECART TYPE σ (µs)	1,164	1,129
MOYENNE $\overline{X}$ (µs)	2,411	1,405

MEDIANE (µs)		2,395	1,134
BORNE MINIMALE $\overline{X} - \sigma$ (µs)		1,612	1,247
BORNE MAXIMALE $\overline{X} + \sigma$ (µs)		3,819	3,575
VARIANCE		1,355	1,275
MINIMUM (µs)		0,006	0,001
MAXIMUM (µs)		10,796	11,485
ETENDUE (µs)		10,791	11,485
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 1{,}794\,\sigma$	$\overline{X} \pm 2{,}740\,\sigma$
	ERREUR DE PRECISION (µs)	2,216	3,094

Variation de la charge du réseau (permutateur D-Link)

TABLEAU II.4 : Sommaire des statistiques de la différence du temps de diffusion dans un réseau Ethernet (selon la charge)

		Aucune charge	Avec charge
ECART TYPE σ		1,129	1,120
MOYENNE $\overline{X}$		1,405	2,636
MEDIANE		1,134	2,588
BORNE MINIMALE $\overline{X} - \sigma$		1,612	1,247
BORNE MAXIMALE $\overline{X} + \sigma$		3,819	3,575
VARIANCE		1,275	1,255
MINIMUM		0,001	0,011
MAXIMUM		11,485	11,771
ETENDUE		11,485	11,761
NIVEAU DE CONFIANCE ≈ 99 %	INTERVALLE DE CONFIANCE	$\overline{X} \pm 2{,}740\,\sigma$	$\overline{X} \pm 2{,}557\,\sigma$
	ERREUR DE PRECISION (µs)	3,094	2,864